

# MATLAB® Report Generator™

User's Guide



# MATLAB®

R2022a



## How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



The MathWorks, Inc.  
1 Apple Hill Drive  
Natick, MA 01760-2098

*MATLAB® Report Generator™ User's Guide*

© COPYRIGHT 1999–2022 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

### Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

### Patents

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

## Revision History

January 1999	First printing	New (Release 11)
December 2000	Second printing	Revised (Release 12)
June 2004	Third printing	Revised for Version 2.02 (Release 14)
August 2004	Online only	Revised for Version 2.1
October 2004	Online only	Revised for Version 2.1.1 (Release 14SP1)
December 2004	Online only	Revised for Version 2.2 (Release 14SP1+)
April 2005	Online only	Revised for Version 2.2.1 (Release 14SP2+)
September 2005	Online only	Revised for Version 2.3.1 (Release 14SP3)
March 2006	Online only	Revised for Version 3.0 (Release 2006a)
September 2006	Online only	Revised for Version 3.1 (Release 2006b)
March 2007	Fourth printing	Revised for Version 3.2 (Release 2007a)
September 2007	Fifth printing	Revised for Version 3.2.1 (Release 2007b)
		This publication was previously for MATLAB® and Simulink®. It is now for MATLAB® only.
March 2008	Online only	Revised for Version 3.3 (Release 2008a)
October 2008	Online only	Revised for Version 3.4 (Release 2008b)
October 2008	Online only	Revised for Version 3.5 (Release 2008b+)
March 2009	Online only	Revised for Version 3.6 (Release 2009a)
September 2009	Online only	Revised for Version 3.7 (Release 2009b)
March 2010	Online only	Revised for Version 3.8 (Release 2010a)
September 2010	Online only	Revised for Version 3.9 (Release 2010b)
April 2011	Online only	Revised for Version 3.10 (Release 2011a)
September 2011	Online only	Revised for Version 3.11 (Release 2011b)
March 2012	Online only	Revised for Version 3.12 (Release 2012a)
September 2012	Online only	Revised for Version 3.13 (Release 2012b)
March 2013	Online only	Revised for Version 3.14 (Release 2013a)
September 2013	Online only	Revised for Version 3.15 (Release 2013b)
March 2014	Online only	Revised for Version 3.16 (Release 2014a)
October 2014	Online only	Revised for Version 4.0 (Release 2014b)
March 2015	Online only	Revised for Version 4.1 (Release 2015a)
September 2015	Online only	Revised for Version 4.2 (Release 2015b)
March 2016	Online only	Revised for Version 5.0 (Release 2016a)
September 2016	Online only	Revised for Version 5.1 (Release 2016b)
March 2017	Online only	Revised for Version 5.2 (Release 2017a)
September 2017	Online only	Revised for Version 5.3 (Release 2017b)
March 2018	Online only	Revised for Version 5.4 (Release 2018a)
September 2018	Online only	Revised for Version 5.5 (Release 2018b)
March 2019	Online only	Revised for Version 5.6 (Release 2019a)
September 2019	Online only	Revised for Version 5.7 (Release 2019b)
March 2020	Online only	Revised for Version 5.8 (Release 2020a)
September 2020	Online only	Revised for Version 5.9 (Release 2020b)
March 2021	Online only	Revised for Version 5.10 (Release 2021a)
September 2021	Online only	Revised for Version 5.11 (Release 2021b)
March 2022	Online only	Revised for Version 5.12 (Release 2022a)





## Getting Started

### 1

<b>MATLAB Report Generator Product Description</b> .....	<b>1-2</b>
Key Features .....	<b>1-2</b>
<b>What Is a Reporter?</b> .....	<b>1-3</b>
Reporters and DOM Objects .....	<b>1-3</b>
Reporter Elements .....	<b>1-4</b>
Using Reporters in a MATLAB Program .....	<b>1-5</b>
<b>Define a New Reporter</b> .....	<b>1-8</b>
Create Class Definition File and Template Copies .....	<b>1-8</b>
Define Fixed and Dynamic Content in the Templates .....	<b>1-9</b>
Define Properties and Specify the Template in the Custom Reporter Class .....	<b>1-10</b>
Use the Custom Reporter .....	<b>1-11</b>
<b>Subclass a Reporter Definition</b> .....	<b>1-13</b>
<b>Java Memory Usage</b> .....	<b>1-14</b>
<b>Working with the Report Explorer</b> .....	<b>1-16</b>
About the Report Explorer .....	<b>1-16</b>
Interactive Report Generation Workflow .....	<b>1-18</b>
<b>Acknowledgments</b> .....	<b>1-19</b>

## Create Your First Report

### 2

<b>Create a Report Generator</b> .....	<b>2-2</b>
<b>Maintain Interactive MATLAB Report</b> .....	<b>2-10</b>
Create a Report Setup File .....	<b>2-10</b>
Add Report Content Using Components .....	<b>2-11</b>
Error Handling for MATLAB Code .....	<b>2-30</b>
Generate a Report .....	<b>2-31</b>

<b>Open a Template File</b> .....	<b>3-2</b>
Open a Word Template File .....	<b>3-2</b>
Open a PDF Template Library File .....	<b>3-3</b>
Open an HTML Template Library File .....	<b>3-4</b>
<b>Reporter Templates</b> .....	<b>3-5</b>
Section Templates .....	<b>3-5</b>
Chapter Templates .....	<b>3-14</b>
<b>Add Report Explorer Content to a Report</b> .....	<b>3-16</b>
<b>Customize Chapters</b> .....	<b>3-17</b>
Customize Static Content .....	<b>3-17</b>
Customize Dynamic Content .....	<b>3-17</b>
Edit a PDF Template .....	<b>3-19</b>
Edit a Word Template .....	<b>3-20</b>
Tips for Editing Headers and Footers in a Word Template .....	<b>3-22</b>
<b>Customize Chapter Page Headers</b> .....	<b>3-24</b>
<b>Create Report API Base Tables</b> .....	<b>3-34</b>
Generate Tables with Numbered Titles .....	<b>3-34</b>
Format BaseTable Tables .....	<b>3-35</b>
Fit Wide BaseTable Tables in a Report .....	<b>3-36</b>
<b>Update a Table of Contents or Generated List in a Word Document</b> ....	<b>3-37</b>
Update a Word Document Programmatically .....	<b>3-37</b>
Update a Word Document Interactively .....	<b>3-37</b>
<b>Table of Contents or Generated List Is Empty in a Word Report</b> .....	<b>3-39</b>
Issue .....	<b>3-39</b>
Possible Solutions .....	<b>3-39</b>
<b>Create a List of Figures or Tables in a Report API Report</b> .....	<b>3-40</b>
Create a List of Figures Section .....	<b>3-40</b>
Create a List of Tables Section .....	<b>3-41</b>
Create a Report That Has a List of Figures and a List of Tables .....	<b>3-41</b>
Customize a List of Figures or Tables Section Title .....	<b>3-44</b>
<b>Create a List of Figures, Tables, or Other Report Elements in a DOM API Report</b> .....	<b>3-45</b>
<b>Create a List of the Captions or Titles of Related Report Elements in a Report API Report</b> .....	<b>3-46</b>
Create a Report That Has a List of Captions Section .....	<b>3-47</b>
Customize a List of Captions Section Title .....	<b>3-48</b>

<b>Report Setups</b> .....	<b>4-2</b>
Setup Hierarchy .....	<b>4-2</b>
Setup Files .....	<b>4-2</b>
Create a Report Setup .....	<b>4-2</b>
<b>Create a New Setup File</b> .....	<b>4-4</b>
Create Setup File Using the Report Explorer .....	<b>4-4</b>
Create Setup File Programmatically .....	<b>4-4</b>
Working with Setup Files .....	<b>4-4</b>
Report Description .....	<b>4-4</b>
<b>Open a Report Setup</b> .....	<b>4-5</b>
Opening a Setup on the MATLAB Path .....	<b>4-5</b>
Opening a Setup Not on the MATLAB Path .....	<b>4-5</b>
Opening a Setup Programmatically .....	<b>4-6</b>
<b>Close a Report Setup</b> .....	<b>4-7</b>
Close a Setup Using the Report Explorer .....	<b>4-7</b>
Close a Setup Programmatically .....	<b>4-7</b>
<b>Save a Report Setup</b> .....	<b>4-8</b>
Save a Setup Under Its Existing Name .....	<b>4-8</b>
Save a Setup Under a New Name .....	<b>4-8</b>
<b>Load Report Setup into MATLAB Workspace</b> .....	<b>4-9</b>
<b>Insert Components</b> .....	<b>4-10</b>
Point-and-Click Method .....	<b>4-10</b>
Drag-and-Drop Method .....	<b>4-10</b>
Fix Context Violations .....	<b>4-10</b>
<b>Set Component Properties</b> .....	<b>4-11</b>
Edit Component Property Values .....	<b>4-11</b>
Computed Property Values .....	<b>4-11</b>
<b>Move Components</b> .....	<b>4-12</b>
Point-and-Click Method .....	<b>4-12</b>
Drag-and-Drop Method .....	<b>4-12</b>
<b>Delete Components</b> .....	<b>4-13</b>
<b>Deactivate Components</b> .....	<b>4-14</b>
<b>Send Components to the MATLAB Workspace</b> .....	<b>4-15</b>

5

<b>Generate a Report</b> .....	<b>5-2</b>
Run a Report .....	5-2
Report Output Options .....	5-2
<b>Select Report Generation Options</b> .....	<b>5-3</b>
Report Options Dialog Box .....	5-3
Location of Report Output File .....	5-3
Report Output Format .....	5-4
PDF Style Sheets .....	5-6
Web Style Sheets .....	5-6
RTF (DSSSL Print) and Word Style Sheets .....	5-7
Report Generation Processing .....	5-7
<b>Report Generation Preferences</b> .....	<b>5-9</b>
Report Generator Preferences Pane .....	5-9
File Format and Extension .....	5-10
Image Formats .....	5-10
Report Viewing .....	5-11
Reset to Defaults .....	5-11
Edit HTML Command .....	5-11
<b>Change Report Locale</b> .....	<b>5-12</b>
<b>Convert XML Documents to Different File Formats</b> .....	<b>5-13</b>
Why Convert XML Documents? .....	5-13
Convert XML Documents Using the Report Explorer .....	5-13
Convert XML Documents Using the Command Line .....	5-14
Edit XML Source Files .....	5-15
<b>Create a Report Log File</b> .....	<b>5-16</b>
<b>Generate MATLAB Code from Report Setup File</b> .....	<b>5-17</b>
<b>Troubleshooting Report Generation Issues</b> .....	<b>5-19</b>
Memory Usage .....	5-19
HTML Report Display on UNIX Systems .....	5-19

**Add Content with Components**

6

<b>Components</b> .....	<b>6-2</b>
Component Formatting .....	6-2
<b>Report Structure Components</b> .....	<b>6-4</b>
<b>Table Formatting Components</b> .....	<b>6-5</b>

<b>Property Table Components</b> .....	<b>6-6</b>
About Property Table Components .....	<b>6-6</b>
Open the Example Report Template .....	<b>6-7</b>
Examine the Property Table Output .....	<b>6-7</b>
Select Object Types .....	<b>6-8</b>
Display Property Name/Property Value Pairs .....	<b>6-8</b>
Edit Table Titles .....	<b>6-10</b>
Enter Text into Table Cells .....	<b>6-10</b>
Add, Replace, and Delete Properties in Tables .....	<b>6-11</b>
Format Table Columns, Rows, and Cells .....	<b>6-12</b>
Zoom and Scroll .....	<b>6-13</b>
Select a Table .....	<b>6-13</b>
<b>Summary Table Components</b> .....	<b>6-14</b>
About Summary Table Components .....	<b>6-14</b>
Open the Example Report Template .....	<b>6-15</b>
Select Object Types .....	<b>6-15</b>
Add and Remove Properties .....	<b>6-15</b>
Set Relative Column Widths .....	<b>6-16</b>
Set Object Row Options .....	<b>6-16</b>
<b>Logical and Looping Components</b> .....	<b>6-17</b>
<b>Edit Figure Loop Components</b> .....	<b>6-18</b>
Figure Loop in a Report .....	<b>6-18</b>
Figure Properties .....	<b>6-19</b>
Loop on the Current Figure .....	<b>6-19</b>
Loop on Visible Figures .....	<b>6-20</b>
Loop on Figures with Tags .....	<b>6-20</b>
Modify Loop Section Options .....	<b>6-20</b>

## Template-Based Report Formatting

# 7

<b>Report Templates</b> .....	<b>7-2</b>
Template-Based Output Types .....	<b>7-2</b>
Templates Versus XSL and DSSSL Style Sheets .....	<b>7-3</b>
Component Styles .....	<b>7-3</b>
Component Templates .....	<b>7-4</b>
Component Holes .....	<b>7-4</b>
Template Cache .....	<b>7-5</b>
<b>Create a Report Template</b> .....	<b>7-6</b>
<b>Copy a Template</b> .....	<b>7-7</b>
<b>Set a Template's Properties</b> .....	<b>7-8</b>
Set Template Properties Interactively .....	<b>7-8</b>
Set Template Properties Programmatically .....	<b>7-8</b>
<b>Open a Template</b> .....	<b>7-9</b>

<b>Generate a Report Using a Template</b> .....	<b>7-10</b>
<b>Default Template Contents</b> .....	<b>7-11</b>
Default Styles .....	7-11
Component Templates .....	7-13
<b>Customize Microsoft Word Report Styles</b> .....	<b>7-16</b>
Customize Default Microsoft Word Component Styles .....	7-16
Create Styles in a Microsoft Word Template .....	7-16
<b>Customize Microsoft Word Component Templates</b> .....	<b>7-18</b>
Custom Word Component Templates .....	7-18
Display the Developer Ribbon in Word .....	7-18
Customize a Word Component Template .....	7-19
Set Default Text Style for a Hole .....	7-19
Distinguish Inline and Block Holes .....	7-21
Avoid Changing Block Holes to Inline Holes .....	7-21
Delete a Hole .....	7-22
Add an Inline Hole .....	7-23
Add a Block Hole .....	7-23
Remove or Modify Chapter Prefix .....	7-23
<b>Customize a Microsoft Word Title Page Template</b> .....	<b>7-25</b>
Create a Custom Template .....	7-25
Change the Color of a Report Title .....	7-25
Assign the Template to a Report .....	7-27
Customize Title Page Content and Layout .....	7-27
<b>Create a Custom HTML or PDF Template</b> .....	<b>7-29</b>
Copy the Template .....	7-29
Edit HTML or PDF Templates .....	7-29
Edit HTML or PDF Styles in a Template .....	7-30
Assign the Template to a Report .....	7-31
Select an HTML Editor .....	7-31

## Create Custom Components

# 8

<b>Create Custom Components</b> .....	<b>8-2</b>
<b>Define Components</b> .....	<b>8-4</b>
Required Component Data .....	8-4
Specify the Location of Component Files .....	8-4
Set Component Display Options .....	8-5
Specify Component Properties .....	8-6
Modify Existing Components .....	8-8
Build Components .....	8-8
Rebuild Existing Components .....	8-8
Remove a Component .....	8-8
<b>Specify Tasks for a Component to Perform</b> .....	<b>8-10</b>
About Component Customization .....	8-10

Required Customization: Specify Format and Content of Report Output . . .	8-10
Change a Component's Outline Text in the Report Explorer Hierarchy . . .	8-12
Modify the Appearance of Properties Dialog Boxes . . . . .	8-12
Specify Additional Component Properties . . . . .	8-13
<b>Define Report Variables . . . . .</b>	<b>8-15</b>

## Create Custom Style Sheets

# 9

<b>Style Sheets . . . . .</b>	<b>9-2</b>
Built-In Versus Custom Style Sheets . . . . .	9-2
Customize Style Sheets Using Data Items . . . . .	9-2
<b>Create a New Style Sheet . . . . .</b>	<b>9-4</b>
<b>Edit, Save, or Delete a Style Sheet . . . . .</b>	<b>9-5</b>
Edit a Style Sheet . . . . .	9-5
Save a Style Sheet . . . . .	9-7
Delete a Style Sheet . . . . .	9-7
<b>Edit Style Sheet Data Items . . . . .</b>	<b>9-8</b>
Data Item Categories in Built-In Style Sheets . . . . .	9-8
Edit Data Items in Simple or Advanced Edit Mode . . . . .	9-11
Data Items . . . . .	9-12
<b>Style Sheet Cells for Headers and Footers . . . . .</b>	<b>9-19</b>
About Style Sheet Cells and Cell Groups . . . . .	9-19
Headers and Footers . . . . .	9-20
Add Content to Headers and Footers Using Templates . . . . .	9-21
Insert Graphics Files . . . . .	9-22
Modify Fonts and Other Properties . . . . .	9-22
<b>Customized Style Sheets . . . . .</b>	<b>9-23</b>
Number Pages in a Report . . . . .	9-23
Add Graphics to Headers in PDF Reports . . . . .	9-24
Change Font Size, Page Orientation, and Paper Type of a Generated Report . . . . .	9-27
Edit Font Size as a Derived Value in XML . . . . .	9-29
<b>Configure PDF Fonts . . . . .</b>	<b>9-32</b>
PDF Font Support for Languages . . . . .	9-32
Identifying When to Specify a Font . . . . .	9-32
Style Sheets Override PDF Font Mapping . . . . .	9-33
Non-English PDF Font Mapping Tasks . . . . .	9-33
lang_font_map.xml File . . . . .	9-33
Locate Non-English Fonts . . . . .	9-35
Add or Modify Language Font Mappings . . . . .	9-36
Specify the Location of Font Files . . . . .	9-36

**10**

**11**

**12**

**13**

<b>Create a Report Program</b> .....	<b>13-3</b>
Required Report Program Tasks and Elements .....	<b>13-3</b>
Optional Report Program Tasks and Elements .....	<b>13-3</b>
Report Generator Program Example .....	<b>13-3</b>
<b>Construct a Report API or DOM API Object</b> .....	<b>13-6</b>
Construct a Report API Object .....	<b>13-6</b>
Construct a DOM API Object .....	<b>13-6</b>
<b>Import the API Packages</b> .....	<b>13-7</b>
<b>Create Report Container</b> .....	<b>13-8</b>
Create Report API Object to Hold Content .....	<b>13-8</b>
Create a DOM Document Object to Hold Content .....	<b>13-8</b>
<b>Add Content to a Report</b> .....	<b>13-10</b>
<b>Add Content as a Group</b> .....	<b>13-12</b>
<b>Output Types and Report Generator Packages</b> .....	<b>13-14</b>
<b>Close a Report</b> .....	<b>13-15</b>
<b>Display Report</b> .....	<b>13-16</b>
<b>Report Formatting Approaches</b> .....	<b>13-17</b>
Style Sheets in Templates .....	<b>13-17</b>
Format Objects .....	<b>13-17</b>
Format Properties .....	<b>13-18</b>
<b>Use Style Sheet Styles</b> .....	<b>13-19</b>
<b>Format Inheritance</b> .....	<b>13-21</b>



<b>Templates for DOM API Report Programs</b> .....	<b>13-22</b>
Template Packages .....	13-22
Styles .....	13-22
Page Layout .....	13-23
Document Part Templates .....	13-23
<b>Form-Based Reporting</b> .....	<b>13-24</b>
<b>Fill the Blanks in a Report Form</b> .....	<b>13-25</b>
<b>Use Subforms in a Report</b> .....	<b>13-26</b>
<b>Create a Microsoft Word Document Part Template Library</b> .....	<b>13-27</b>
Create Document Part Template Library in a Word Template .....	13-27
Word Document Part List Limitations .....	13-28
<b>Create an HTML Document Part Template Library</b> .....	<b>13-30</b>
HTML Document Part Template Library Structure .....	13-30
<b>Create a PDF Document Part Template Library</b> .....	<b>13-32</b>
PDF Document Part Template Library Structure .....	13-32
Document Part Template Library Contents .....	13-33
<b>Simplify Filling in Forms</b> .....	<b>13-37</b>
<b>Create and Format Text</b> .....	<b>13-39</b>
Create Text .....	13-39
Create Special Characters .....	13-39
Append HTML or XML Markup .....	13-39
Format Text .....	13-40
<b>Format Numbers</b> .....	<b>13-43</b>
Set the Default Number Format .....	13-43
Specify the Format for One Number .....	13-43
Specify the Number Format for a Paragraph, Table, or List .....	13-44
<b>Create and Format Paragraphs</b> .....	<b>13-46</b>
Create a Paragraph .....	13-46
Create a Heading .....	13-46
Format a Paragraph .....	13-46
<b>Create and Format Lists</b> .....	<b>13-51</b>
Create a List from a MATLAB Array .....	13-51
Create a List One Item at a Time .....	13-53
Create a Multilevel List .....	13-54
Format a List Using Template-Defined Styles .....	13-57
Format a List Programmatically .....	13-58
Format a List Item in a Multilevel List .....	13-59
<b>Choose Type of Table to Create</b> .....	<b>13-62</b>
<b>Create Informal Tables</b> .....	<b>13-64</b>
Create an Informal Table from a MATLAB Array .....	13-64
Build an Informal Table from Rows and Entries .....	13-65

<b>Create Formal Tables</b> .....	<b>13-66</b>
Create a Formal Table from MATLAB Arrays .....	<b>13-66</b>
Create a Formal Table from Rows and Entries .....	<b>13-66</b>
<b>Create Tables from MATLAB Tables</b> .....	<b>13-68</b>
Create a Table from a MATLAB Table .....	<b>13-68</b>
Format a Table Created from a MATLAB Table .....	<b>13-69</b>
<b>Format Tables</b> .....	<b>13-72</b>
Format Tables Using Template-Defined Styles .....	<b>13-72</b>
Format Tables Programmatically .....	<b>13-75</b>
<b>Create Links</b> .....	<b>13-82</b>
Create an Internal Link .....	<b>13-82</b>
Create an External Link .....	<b>13-82</b>
Add Text or Images to Links .....	<b>13-83</b>
Create a Page Reference .....	<b>13-83</b>
<b>Create a Dynamic Table</b> .....	<b>13-85</b>
Create Dynamic Table From Table Objects .....	<b>13-85</b>
Create Dynamic Table Using Table Constructor .....	<b>13-86</b>
<b>Create and Format Images</b> .....	<b>13-88</b>
Create an Image .....	<b>13-88</b>
Resize an Image .....	<b>13-88</b>
Image Storage .....	<b>13-89</b>
Links from an Image .....	<b>13-89</b>
<b>Create a Title Page</b> .....	<b>13-90</b>
<b>Create a Table of Contents</b> .....	<b>13-92</b>
Create a TOC in Your Report Program .....	<b>13-92</b>
Use a Template to Create a Microsoft Word TOC .....	<b>13-93</b>
Create Table of Contents in HTML or PDF Templates .....	<b>13-95</b>
Set Outline Levels of Section Heads .....	<b>13-96</b>
<b>Create Image Maps</b> .....	<b>13-99</b>
<b>Automatically Number Document Content</b> .....	<b>13-101</b>
Automatically Number Content Programmatically .....	<b>13-101</b>
Automatically Number Content Using Part Templates .....	<b>13-102</b>
<b>Convert HTML Content to a DOM Object</b> .....	<b>13-105</b>
Prepare HTML Before Conversion .....	<b>13-105</b>
Convert HTML Content in a String .....	<b>13-106</b>
Convert HTML File Content .....	<b>13-106</b>
<b>Prepare HTML for Conversion to a DOM Object</b> .....	<b>13-108</b>
<b>Requirements for Converting HTML to a DOM Object</b> .....	<b>13-110</b>
Supported HTML Elements and Attributes .....	<b>13-110</b>
Supported HTML CSS Style Attributes for All Elements .....	<b>13-112</b>
Support for HTML Character Entities .....	<b>13-113</b>
DOCTYPE Declaration .....	<b>13-113</b>

<b>Display Progress and Debugger Messages</b> .....	<b>13-115</b>
Report Generation Messages .....	<b>13-115</b>
Display DOM Default Messages .....	<b>13-115</b>
Create and Display a Progress Message .....	<b>13-116</b>
<b>Create a Standalone Application from a Report Program</b> .....	<b>13-118</b>
<b>Create a Microsoft Word Template</b> .....	<b>13-119</b>
<b>Add Holes in a Microsoft Word Template</b> .....	<b>13-120</b>
Open Word Template .....	<b>13-120</b>
Display Formatting Symbols .....	<b>13-120</b>
Display the Developer Tab .....	<b>13-120</b>
Add a Hole Using a Rich Text Content Control .....	<b>13-121</b>
Specify a Default Style for Text That Fills a Hole .....	<b>13-124</b>
<b>Modify Styles in a Microsoft Word Template</b> .....	<b>13-126</b>
Edit Styles in a Word Template .....	<b>13-126</b>
Add Styles to a Word Template .....	<b>13-127</b>
<b>Create an HTML or PDF Template</b> .....	<b>13-130</b>
Edit an HTML or PDF Template .....	<b>13-130</b>
<b>Add Holes in HTML and PDF Templates</b> .....	<b>13-132</b>
Types of Holes .....	<b>13-132</b>
Create a Hole .....	<b>13-132</b>
<b>PDF and HTML Document Parts and Holes</b> .....	<b>13-134</b>
Add Template to PDF Document Part Template Library .....	<b>13-134</b>
Use the Document Part Template in a Report Program .....	<b>13-135</b>
<b>Modify Styles in HTML Templates</b> .....	<b>13-137</b>
<b>Modify Styles in PDF Templates</b> .....	<b>13-138</b>
PDF Style Sheets .....	<b>13-138</b>
Hyphenation Styles in PDF Templates .....	<b>13-141</b>
<b>Create Chapters</b> .....	<b>13-143</b>
<b>Create Page Layout Sections</b> .....	<b>13-144</b>
Define Page Layouts in a Word Template .....	<b>13-144</b>
Define Page Layouts in a PDF Template .....	<b>13-144</b>
Watermarks in PDF Page Layouts .....	<b>13-145</b>
Navigate Template-Defined Page Layouts .....	<b>13-146</b>
Override Template Page Layouts in Your Report Program .....	<b>13-146</b>
Create Layouts Programmatically .....	<b>13-147</b>
<b>Create Page Footers and Headers</b> .....	<b>13-148</b>
Use Page Headers and Footers in a Template .....	<b>13-148</b>
Create Running Page Headers and Footers .....	<b>13-151</b>
Create Page Headers and Footers Programmatically .....	<b>13-153</b>
<b>Add Complex Page Numbers in Microsoft Word</b> .....	<b>13-155</b>
<b>Functional Report</b> .....	<b>13-157</b>

<b>Object-Oriented Report</b> .....	<b>13-160</b>
<b>Report Formatting</b> .....	<b>13-163</b>
<b>Create and Use a Custom Finder</b> .....	<b>13-164</b>
<b>Resolve Error from Closing a PDF Document on a Cloud Drive</b> .....	<b>13-188</b>
Issue .....	<b>13-188</b>
Possible Solutions .....	<b>13-188</b>

## Programmatic PowerPoint Presentation Creation

# 14

<b>Create a Presentation Generator</b> .....	<b>14-2</b>
Update Presentation Content .....	<b>14-3</b>
Two Ways to Use the PPT API .....	<b>14-5</b>
PPT API Applications and PowerPoint Templates .....	<b>14-5</b>
Template Elements .....	<b>14-5</b>
<b>Create PPT Objects</b> .....	<b>14-7</b>
PPT Objects .....	<b>14-7</b>
Use a PPT Constructor .....	<b>14-7</b>
PPT Objects Created Without Constructors .....	<b>14-7</b>
<b>Import the PPT API Package</b> .....	<b>14-9</b>
<b>Get and Set PPT Object Properties</b> .....	<b>14-10</b>
<b>Create a Presentation Object to Hold Content</b> .....	<b>14-12</b>
<b>Generate a Presentation</b> .....	<b>14-13</b>
<b>Display Presentation Generation Messages</b> .....	<b>14-14</b>
Presentation Generation Messages .....	<b>14-14</b>
Display PPT Default Messages .....	<b>14-14</b>
Create and Display a Progress Message .....	<b>14-15</b>
<b>Create a Standalone Application from a Presentation Program</b> .....	<b>14-17</b>
<b>Presentation Formatting Approaches</b> .....	<b>14-18</b>
Template Formatting .....	<b>14-18</b>
Format Objects .....	<b>14-19</b>
Format Properties .....	<b>14-19</b>
Interactive Formatting of Slide Content .....	<b>14-19</b>
<b>Presentation Format Inheritance</b> .....	<b>14-21</b>
<b>Set Up a PowerPoint Presentation Template</b> .....	<b>14-23</b>
Specify a Template .....	<b>14-23</b>
Copy the Default Template .....	<b>14-23</b>
Customize a Template Using the PPT API .....	<b>14-23</b>
Customize a Template Interactively in PowerPoint .....	<b>14-24</b>

<b>Access PowerPoint Template Elements</b> .....	<b>14-32</b>
PPT API Applications and PowerPoint Templates .....	<b>14-32</b>
Template Elements .....	<b>14-32</b>
View and Change Slide Master Names .....	<b>14-33</b>
View and Change Slide Layout Names .....	<b>14-33</b>
View and Change Placeholder and Content Object Names .....	<b>14-34</b>
<b>Define a Style Using Format Objects</b> .....	<b>14-37</b>
<b>Use Format Properties</b> .....	<b>14-39</b>
Dot Notation .....	<b>14-39</b>
Get the Properties of an Object .....	<b>14-39</b>
Set the Properties of an Object .....	<b>14-40</b>
<b>Update Presentation Content Programmatically</b> .....	<b>14-41</b>
Generate the Existing Presentation .....	<b>14-41</b>
Updates to the Presentation .....	<b>14-42</b>
Set Up the Existing Presentation .....	<b>14-43</b>
Import the PPT API Package .....	<b>14-44</b>
Create the Presentation Object .....	<b>14-44</b>
Replace a Picture .....	<b>14-45</b>
Replace Text with Links .....	<b>14-45</b>
Replace a Table .....	<b>14-45</b>
Insert a New Slide .....	<b>14-46</b>
Generate and View the Presentation .....	<b>14-46</b>
Code for myUpdatedPresentation .....	<b>14-46</b>
<b>Create a Presentation Programmatically</b> .....	<b>14-49</b>
<b>Add Slides</b> .....	<b>14-56</b>
Specify the Order of a Slide .....	<b>14-56</b>
Specify the Slide Master .....	<b>14-57</b>
<b>Add and Replace Presentation Content</b> .....	<b>14-58</b>
Set Up the Template .....	<b>14-58</b>
Replace Content .....	<b>14-58</b>
Add and Replace Text .....	<b>14-59</b>
Add or Replace a Table .....	<b>14-61</b>
Add or Replace a Picture .....	<b>14-62</b>
<b>Create and Format Text</b> .....	<b>14-65</b>
Create Text .....	<b>14-65</b>
Create a Subscript or Superscript .....	<b>14-65</b>
Format Text .....	<b>14-65</b>
<b>Create and Format Paragraphs</b> .....	<b>14-67</b>
Create a Paragraph .....	<b>14-67</b>
Format Paragraph Content .....	<b>14-67</b>
<b>Create and Format Tables</b> .....	<b>14-69</b>
Create a Table .....	<b>14-69</b>
Format a Table .....	<b>14-69</b>
View Table Style Names .....	<b>14-73</b>

<b>Create and Format Pictures</b> .....	<b>14-76</b>
Create a Picture .....	<b>14-76</b>
Format a Picture .....	<b>14-76</b>
<b>Create and Format Links</b> .....	<b>14-78</b>
Create an Internal Link .....	<b>14-78</b>
Create an External Link .....	<b>14-82</b>
Format an Internal or External Link .....	<b>14-82</b>
<b>Generate a Presentation From the Results of a MATLAB Application</b> .	<b>14-84</b>

## Classes Being Removed

**15**

## Form-Based Reports

**16**

<b>Form-Based Reports</b> .....	<b>16-2</b>
Workflow for Creating Form-Based Reports .....	<b>16-2</b>
Create Multiform-Based Report Setups .....	<b>16-3</b>
Define Page Layouts in a Form-Based Report Setup .....	<b>16-4</b>
<b>Create a Simple Form-Based Setup</b> .....	<b>16-6</b>
Create a Word Template .....	<b>16-6</b>
Create the Report Setup File .....	<b>16-10</b>
Generate the Report .....	<b>16-12</b>
<b>Report Form</b> .....	<b>16-13</b>
Report File Location Options .....	<b>16-13</b>
Report Output Format .....	<b>16-14</b>
Report Generation Processing .....	<b>16-15</b>
Report Description .....	<b>16-16</b>

## MATLAB Report Generator Task Examples

**17**

<b>Specify Space Between Paragraphs</b> .....	<b>17-3</b>
<b>Side-by-Side Tables</b> .....	<b>17-6</b>
<b>Fit Wide Tables in a Page</b> .....	<b>17-8</b>
<b>Span a Table Entry Across Rows and Columns</b> .....	<b>17-13</b>
<b>Side-by-Side Images</b> .....	<b>17-20</b>

<b>Side-by-Side Figures</b> .....	<b>17-22</b>
<b>Scale Image to Fit Page</b> .....	<b>17-24</b>
<b>Hyperlink Image</b> .....	<b>17-26</b>
<b>Create a Report with Landscape Pages</b> .....	<b>17-30</b>
<b>Create a Report with Portrait and Landscape Pages</b> .....	<b>17-38</b>
<b>Set Table Column Width</b> .....	<b>17-43</b>
<b>Number Section Headings, Table Titles, and Figure Captions Programmatically</b> .....	<b>17-46</b>
<b>Align Table Entry Content Horizontally</b> .....	<b>17-51</b>
<b>Create a Zebra-Striped Table</b> .....	<b>17-54</b>
<b>Set Page Margins in a Word Report</b> .....	<b>17-59</b>
<b>Set Page Margins in a PDF Report</b> .....	<b>17-66</b>
<b>Programmatically Number Pages</b> .....	<b>17-73</b>
<b>Create an Inline Equation in a Report</b> .....	<b>17-80</b>
<b>Custom Styled Word List</b> .....	<b>17-82</b>
<b>Multilevel List</b> .....	<b>17-86</b>
<b>Number Pages in a PDF Template</b> .....	<b>17-92</b>
<b>Number Pages in a Word Template</b> .....	<b>17-95</b>
<b>Excel to PDF</b> .....	<b>17-99</b>
<b>Prevent MATLAB Figure Display During Report Generation</b> .....	<b>17-106</b>
<b>Create a Table from a Cell Array</b> .....	<b>17-109</b>
<b>Format Numbers in a Table</b> .....	<b>17-112</b>
<b>Center Figure Snapshot on a Page</b> .....	<b>17-114</b>
<b>Center Image on a Page</b> .....	<b>17-121</b>
<b>Center Formal Image in a PDF Report</b> .....	<b>17-126</b>
<b>Define Styles Programmatically</b> .....	<b>17-131</b>
<b>Customize the Page Size and Margins of a Word Report Using Templates</b> .....	<b>17-135</b>

<b>Customize the Page Size and Margins of a PDF Report Using Templates</b> .....	<b>17-145</b>
<b>Customize the Page Size and Margins of a Report Programmatically</b>	<b>17-154</b>
<b>Generate a Microsoft Word Document with Page Borders</b> .....	<b>17-157</b>
<b>Generate a PDF Document with Page Borders</b> .....	<b>17-159</b>
<b>Generate a Report API Report with Page Borders</b> .....	<b>17-161</b>



# Getting Started

---

- “MATLAB Report Generator Product Description” on page 1-2
- “What Is a Reporter?” on page 1-3
- “Define a New Reporter” on page 1-8
- “Subclass a Reporter Definition” on page 1-13
- “Java Memory Usage” on page 1-14
- “Working with the Report Explorer” on page 1-16
- “Acknowledgments” on page 1-19

## **MATLAB Report Generator Product Description**

### **Design and automatically generate reports from MATLAB applications**

MATLAB Report Generator provides functions and APIs that integrate reporting capabilities into MATLAB applications. You can develop programs that generate reports in PDF, Microsoft® Word, Microsoft PowerPoint®, and HTML. MATLAB Report Generator enables you to dynamically capture results and figures from your MATLAB code and document those results in a single report that can be shared with others in your organization. You can use the prebuilt, customizable Word and HTML templates or design reports based on your organization's templates and standards.

### **Key Features**

- Automated reporting from MATLAB
- PDF, Microsoft Word, Microsoft PowerPoint, and HTML formats
- Templates for programmatic reporting
- Comprehensive APIs for creating scalable report generators

## What Is a Reporter?

Reporters are MATLAB objects that generate formatted content when added to a MATLAB Report Generator Report object. MATLAB Report Generator provides reporters for generating common report components, such as title pages, tables of contents, chapters, subsections, figures, and MATLAB variables values. You can customize the content and appearance of these reporters. You can also create your own reporters. For a list of built-in Report API objects, enter this MATLAB command:

```
help mlreportgen.report
```

### Reporters and DOM Objects

In addition to reporters, MATLAB Report Generator provides another set of objects for generating report content. These objects are Document Object Model (DOM) objects. They implement a model of a document used by HTML, Word, and other document creation software. The model defines a document as a hierarchy of objects commonly found in documents, such as text strings, paragraphs, images, and tables. The DOM API contains software objects that generate these basic document objects. For a list of the DOM objects, enter this MATLAB command:

```
help mlreportgen.dom
```

Reporters, by contrast, create high-level document structures, such as title pages, tables of contents and chapters, that occur in many, but not all types of documents. The advantage of reporters is that a single reporter can create content that would require many DOM objects. However, a report generator program typically requires both DOM and reporter objects. For example, a chapter reporter generates the title and page layout of a report chapter, but not its content. The DOM API provides text, paragraph, table, list, image, and other objects that you can use to create reporter content.

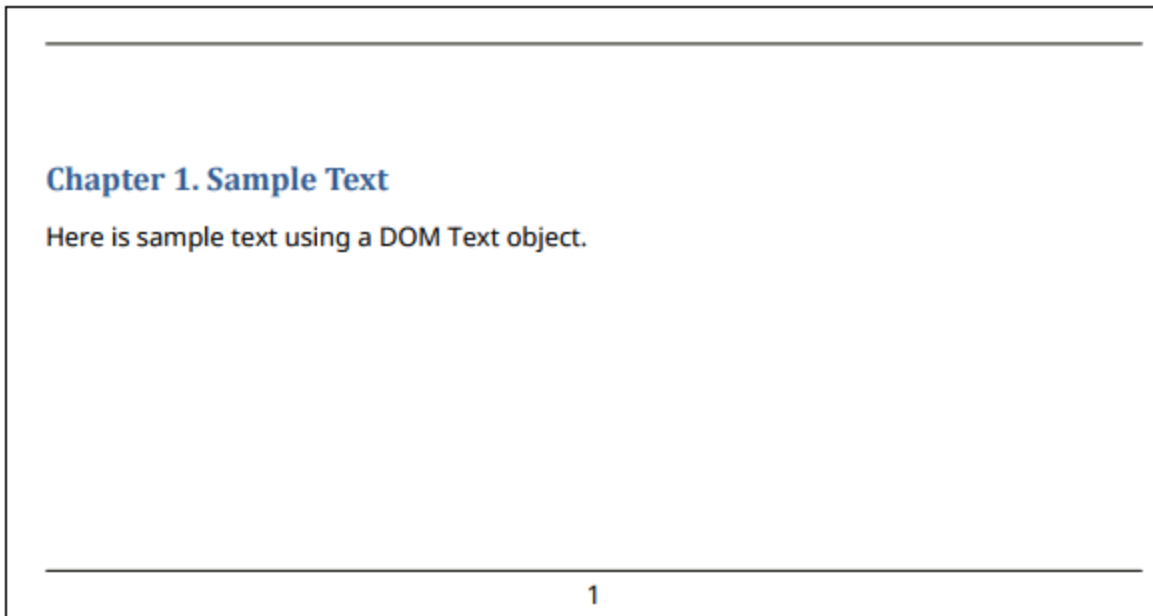
The following MATLAB program illustrates using both reporters and DOM objects to create a PDF report. The program uses a DOM Text object to add a block of text to the chapter. All other objects in this example (Report, TitlePage, TableOfContents, and Chapter) are reporter objects.

```
rpt = mlreportgen.report.Report('myreport','pdf');
append(rpt,mlreportgen.report.TitlePage('Title','My Report',...
    'Author','Myself'))
append(rpt,mlreportgen.report.TableOfContents)
ch = mlreportgen.report.Chapter('Title','Sample Text');
append(ch,mlreportgen.dom.Text...
    ('Here is sample text using a DOM Text object.'))
append(rpt,ch)
close(rpt)
rptview(rpt)
```



**Table of Contents**

[Chapter 1. Sample Text](#)..... 1



## Reporter Elements

A reporter typically includes the following elements:

- Template documents that define the appearance, fixed content, and holes for dynamic content generated by the reporter. A reporter typically provides a set of templates files, one for each

supported output type: Word, PDF, and HTML. Each template file contains a library of templates used by the reporter to format its content. For example, the Report API `TitlePage` reporter uses a template named `TitlePage` to format a title page. The `TitlePage` template is stored in the template libraries of its template files. You can modify this template to rearrange or add content to a title page. For information, see “Templates”.

- Properties that specify the dynamic content generated by the reporter. These properties correspond to holes in the reporter template. A reporter fills the template holes with the values of the corresponding properties.
- MATLAB class that defines the reporter properties and methods you use to create and manipulate the reporter. Reporter class names begin with the prefix, `mlreportgen.report`. For example, the title page reporter is `mlreportgen.report.TitlePage`. You can omit the prefix in a MATLAB script or function by inserting this statement at the beginning of the script or function:

```
import mlreportgen.report.*
```

Likewise, you can include `import mlreportgen.dom.*` to use short DOM class names.

- Constructor method that creates a reporter object as an instance of the reporter class. The name of the constructor is the same as the name of the class.
- DOM object that contains the content generated by the report. This object is referred to as the implementation of the reporter. Each reporter has a `getImpl` method that creates the implementation object, which is typically a `DOM DocumentPart` object.

## Using Reporters in a MATLAB Program

To generate content in a report program, follow these steps:

- 1 “Create a Report Object” on page 1-5
- 2 “Create an Instance of the Reporter” on page 1-5
- 3 “Set the Properties of an Existing Reporter” on page 1-6
- 4 “Add the Reporter to a Report” on page 1-6
- 5 “Close the Report Object” on page 1-6

The example program described in these steps creates a simple document that includes only a title page. However, the steps demonstrate the tasks to create a full report. The full program listing is shown after the step descriptions.

### Create a Report Object

Create a Report object (`mlreportgen.report.Report`) to contain the content generated by the report. The report object uses a `DOM Document` object to hold content generated by reporters added to the report. This code imports the Report API package, which enables the code to use short class names. Then, it creates a PDF report object (`rpt`).

```
import mlreportgen.report.*
rpt = Report('myReport', 'pdf');
```

### Create an Instance of the Reporter

Create an instance of the reporter class, that is, instantiate the reporter, using its constructor. The constructor can also set the properties of the reporter object it creates. For example, this code creates a title page reporter (`tp`) and sets its `Title` and `Author` properties.

```
tp = TitlePage('Title', 'My Report', 'Author', 'John Smith');
```

### **Set the Properties of an Existing Reporter**

To set reporter properties after a program has created a reporter, the program can use MATLAB dot notation. For example, this code sets the `Subtitle` and `PubDate` properties of a `TitlePage` reporter (`tp`).

```
tp.Subtitle = 'on My Project';  
tp.PubDate = date;
```

### **Add the Reporter to a Report**

To generate content using a reporter, a report program must add the reporter to the report object, using the `append` method of the report object. The `append` method works by invoking the `getImpl` method of that reporter. The `getImpl` method creates the implementation of the reporter. Then, the `append` method adds the implementation to the DOM Document object that serves as the implementation of the report object. You can also use the `append` method to add DOM objects to the report. You cannot, however, add another DOM Document to a report.

For example, this code adds the title page reporter (`tp`) to the report (`rpt`).

```
append(rpt, tp)
```

### **Close the Report Object**

When a report program has finished adding content to a report, it must close the report, using the `close` method of the report object. Closing a report writes the report content to a document file of the type, such as PDF, specified by the constructor of the report object.

```
close(rpt)
```

This code is the complete program for the report, which includes only a title page.

```
import mlreportgen.report.*  
  
rpt = Report('myReport', 'pdf');  
  
tp = TitlePage('Title', 'My Report', ...  
    'Author', 'John Smith');  
tp.Subtitle = 'on My Project';  
tp.PubDate = date;  
  
append(rpt, tp)  
close(rpt)  
rptview(rpt)
```



### **See Also**

`mlreportgen.report.TitlePage` | `mlreportgen.report.Report` |  
`mlreportgen.report.TableOfContents` | `mlreportgen.dom.Text`

### **More About**

- “Create a Report Program” on page 13-3
- “Define a New Reporter” on page 1-8
- “Subclass a Reporter Definition” on page 1-13

## Define a New Reporter

MATLAB Report Generator allows you to define a custom reporter based on the `mlreportgen.report.Reporter` class or a built-in Report API reporter. Base your reporter on a built-in reporter when the built-in reporter meets most of your requirements and you want to rearrange or expand the reporter content. See “Subclass a Reporter Definition” on page 1-13. Base your reporter on the `mlreportgen.report.Reporter` class to define an entirely new reporter.

To create a reporter based on `mlreportgen.report.Reporter`, use the `mlreportgen.report.Reporter.customizeReporter` method. The method creates a skeleton class definition file and makes copies of the default template files for each report output type.

To complete the custom reporter definition:

- In the template file for each report output type supported by the reporter, define the fixed content and holes for the dynamic content that the reporter generates.
- In the custom reporter class, define properties that correspond to the template holes.

To use the reporter in a Report API report, create an object of the reporter class, set the property values, and append the object to the report.

## Create Class Definition File and Template Copies

Create a class definition file for a custom reporter class and make copies of the default templates by calling the `mlreportgen.report.Reporter.customizeReporter` method. Provide the path and name of the class to be created as the input argument to the method. To create the reporter class in a class folder, precede the class name with the `@` character.

For example, this code creates a class file `MyTitlePage.m` in the folder named `@MyTitlePage`:

```
mlreportgen.report.Reporter.customizeReporter("@MyTitlePage")
```

The class definition file `MyTitlePage.m` contains:

```
classdef MyTitlePage < mlreportgen.report.Reporter

    properties
    end

    methods
        function obj = MyTitlePage(varargin)
            obj = obj@mlreportgen.report.Reporter(varargin{:});
        end
    end

    methods (Hidden)
        function templatePath = getDefaultTemplatePath(~, rpt)
            path = MyTitlePage.getClassFolder();
            templatePath = ...
                mlreportgen.report.ReportForm.getFormTemplatePath(...
                    path, rpt.Type);
        end
    end
end
```



```

methods (Static)
    function path = getClassFolder()
        [path] = fileparts(mfilename('fullpath'));
    end

    function createTemplate(templatePath, type)
        path = MyTitlePage.getClassFolder();
        mlreportgen.report.ReportForm.createFormTemplate(...
            templatePath, type, path);
    end

    function customizeReporter(toClasspath)
        mlreportgen.report.ReportForm.customizeClass(...
            toClasspath, "MyTitlePage");
    end
end
end

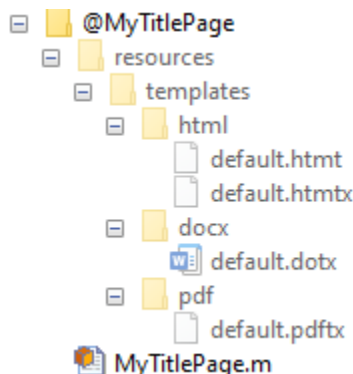
```

The class includes a constructor and the hidden method `getDefaultTemplatePath`. The base reporter class uses the `getDefaultTemplatePath` method to retrieve the `MyTitlePage` reporter template that corresponds to the output type of the report to which the reporter is added. For example, if you add the reporter to an `mlreportgen.report.Report` object whose output type is PDF, the base reporter class returns the path of the PDF template that resides in the `resources/templates/pdf` subfolder of your reporter definition folder.

The `mlreportgen.report.Reporter.customizeReporter` method stores copies of the default template files for each report output type in the `resources/templates` subfolder of the folder that contains the class definition file. The paths of the template files relative to `resources/templates` are:

- `docx/default.dotx`
- `pdf/default.pdfxtx`
- `html/default.htmlt`
- `html/default.htmltx`

For example, the `@MyTitlePage` folder has this structure:



## Define Fixed and Dynamic Content in the Templates

Customize the template files by defining the fixed content and holes for the dynamic content that the custom reporter generates. You only need to customize the template files for the report output types

supported by the custom reporter. For example, if the reporter supports only Word reports, customize only the `dotx` template file.

If the custom reporter requires multiple templates, store the templates in a template library in the template file. If the reporter requires only one template, you can store the template content in the body of the template file or in a template in the template library. For example, for the `MyTitlePage` reporter, which requires only one template, you can add the template fixed content and holes for dynamic content to the body of the template file or to an entry named `MyTitlePage` in the template file library.

Define the styles used by the template in the style sheet of the template file.

See “Create a Microsoft Word Document Part Template Library” on page 13-27, “Create a PDF Document Part Template Library” on page 13-32, and “Create an HTML Document Part Template Library” on page 13-30.

## Define Properties and Specify the Template in the Custom Reporter Class

In the custom reporter class:

- Define a property for each of the holes defined by the templates, including holes in the headers and footers. The property corresponding to a hole must have the same name as the hole. For example, if the reporter templates define a hole named `Title`, the class definition file must define a property named `Title`.
- If the reporter uses a template stored in the template library of the template file, add a line to the constructor that sets the property `TemplateName` to the name of the reporter template. You do not have to specify this property in the class definition file because the customized class inherits this property from the base `mlreportgen.report.Reporter` class.

For example, this class definition file defines `Title`, `Author`, and `Version` properties and specifies that the template name is `MyTitlePage`

```
classdef MyTitlePage < mlreportgen.report.Reporter

    properties
        Title = "";
        Author = "";
        Version = "";
    end

    methods
        function obj = MyTitlePage(varargin)
            obj = obj@mlreportgen.report.Reporter(varargin{:});
            obj.TemplateName = "MyTitlePage";
        end
    end

    methods (Hidden)
        function templatePath = getDefaultTemplatePath(~, rpt)
            path = MyTitlePage.getClassFolder();
            templatePath = ...
                mlreportgen.report.ReportForm.getFormTemplatePath(...
                    path, rpt.Type);
        end
    end
end
```

```

        end
    end
    methods (Static)
        function path = getClassFolder()
            [path] = fileparts(mfilename('fullpath'));
        end

        function createTemplate(templatePath, type)
            path = MyTitlePage.getClassFolder();
            mlreportgen.report.ReportForm.createFormTemplate(...
                templatePath, type, path);
        end

        function customizeReporter(toClasspath)
            mlreportgen.report.ReportForm.customizeClass(...
                toClasspath, "MyTitlePage");
        end
    end
end
end

```

## Use the Custom Reporter

To use your custom reporter:

- 1 Create an object of the reporter class.
- 2 Set the values of the properties that you defined in the class.
- 3 Append the object to a report.

For example:

```

import mlreportgen.report.*

rpt = Report("myreport", "pdf");

titlePage = MyTitlePage;
titlePage.Title = "My Report";
titlePage.Author = "Me";
titlePage.Version = "1.0"
append(rpt, titlePage);

close(rpt);
rptview(rpt);

```

## See Also

mlreportgen.report.TitlePage | mlreportgen.report.Report |  
mlreportgen.report.TableOfContents | mlreportgen.dom.Text |  
mlreportgen.dom.Document

## More About

- “Create a Microsoft Word Document Part Template Library” on page 13-27

- “Create an HTML Document Part Template Library” on page 13-30
- “Create a PDF Document Part Template Library” on page 13-32
- “What Is a Reporter?” on page 1-3
- “Subclass a Reporter Definition” on page 1-13

## Subclass a Reporter Definition

If a built-in reporter meets some of your requirements, consider subclassing the reporter. Subclassing a reporter allows you to rearrange and expand the content of a built-in reporter. To subclass a built-in reporter:

- 1 Create a custom reporter definition based on the class definition of the built-in reporter, using the `customizeReporter` method of the built-in reporter.

For example, the following code creates a folder named `@MyTitlePage` in the current folder.

```
mlreportgen.report.TitlePage.customizeReporter('@MyTitlePage');
```

The created folder contains a class definition file named `MyTitlePage.m`. The class definition file defines a subclass of the built-in `TitlePage` reporter. The class definition folder also contains copies of the `TitlePage` reporter templates, which are stored in a subfolder named `resources`.

- 2 Edit the template copies to rearrange the holes for the content of the built-in reporter or add holes for additional generated content.
- 3 If you add holes to the templates of the new reporter, edit the reporter class definition file to define properties that specify the content that fills the holes. Define a property for each hole that you have added to the template of the new reporter. The name of the property must be the same as the name of the hole.

### See Also

`mlreportgen.report.TitlePage`

### More About

- “What Is a Reporter?” on page 1-3
- “Define a New Reporter” on page 1-8

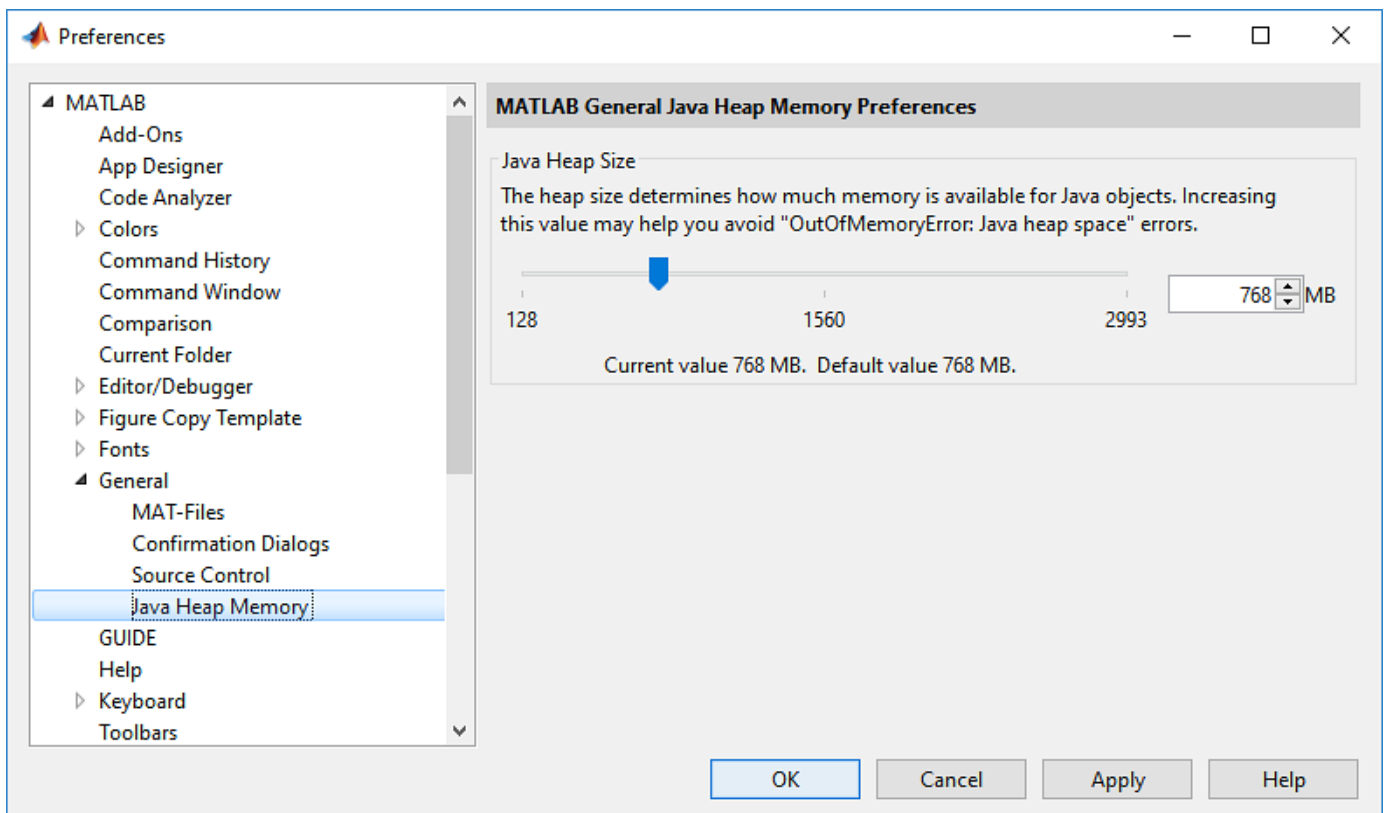
## Java Memory Usage

MATLAB Report Generator uses Java heap memory to generate PDF documents. The default Java heap size that MATLAB allocates may not be enough to convert large PDF documents. The error that occurs if your report conversion runs out of Java heap memory is:

```
Document conversion failed. Java exception occurred:  
java.lang.OutOfMemoryError
```

To generate your PDF document, increase the Java heap space to the maximum amount:

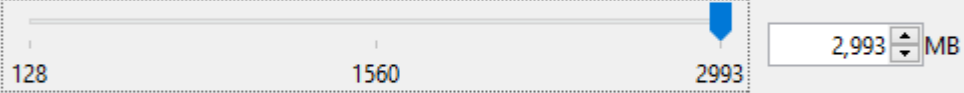
- 1 Click **Preferences** on the MATLAB toolbar.
- 2 Expand **General**
- 3 To open the MATLAB General Java Heap memory Preferences panel, click **Java Heap Memory**



- 4 Move the slider all the way to the right to set the Java Heap Size to the maximum value.


**Java Heap Size**

The heap size determines how much memory is available for Java objects. Increasing this value may help you avoid "OutOfMemoryError: Java heap space" errors.



128 1560 2993 2,993 MB

Current value 768 MB. Default value 768 MB.

 You must restart MATLAB for the above change to take effect.

- 5 To enable the new Java heap size, restart MATLAB.

## Working with the Report Explorer

In this section...
“About the Report Explorer” on page 1-16
“Interactive Report Generation Workflow” on page 1-18

### About the Report Explorer

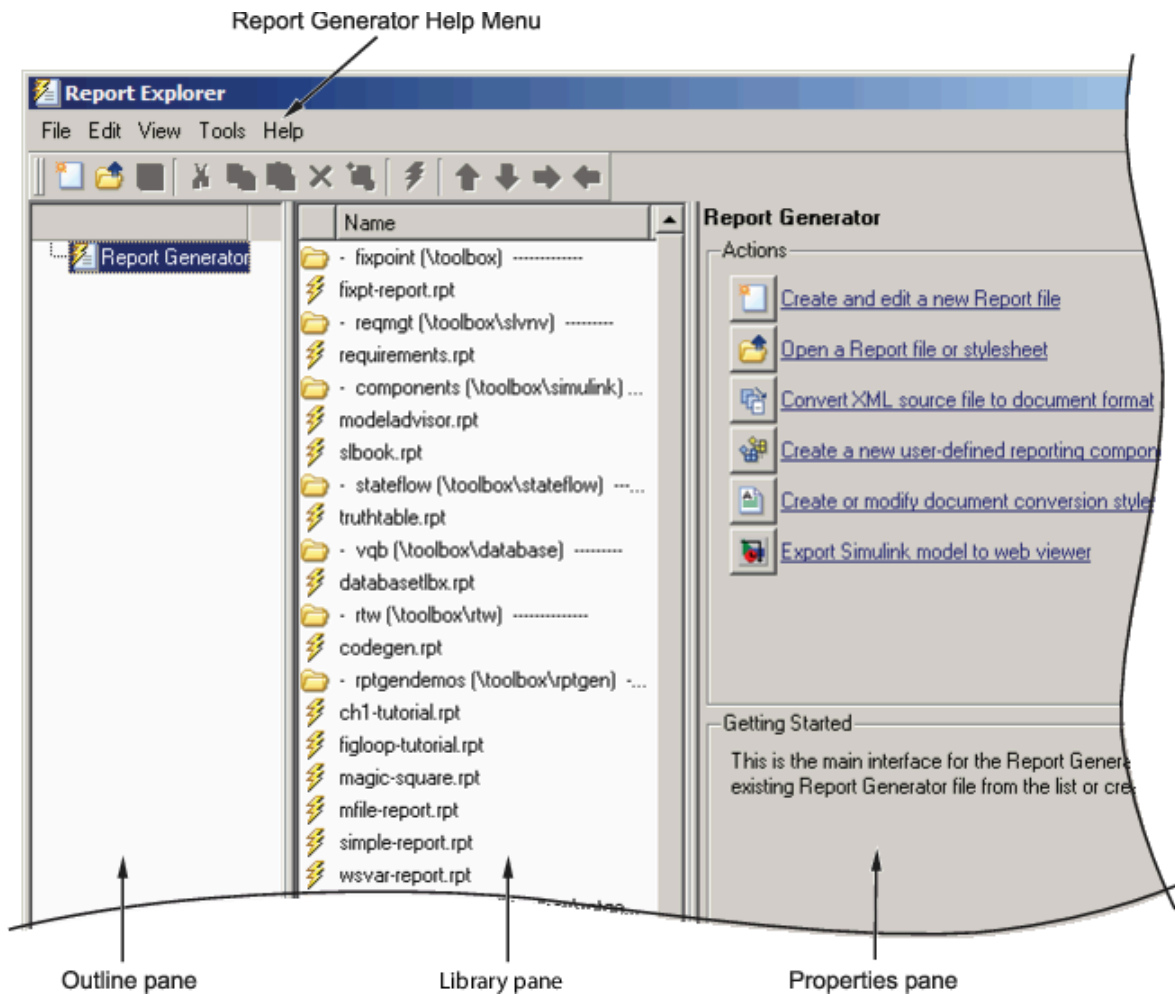
Use the *Report Explorer* to:

- Create and modify report setup files.
- Apply style sheets to format the generated report.
- Specify the report file format.
- Generate reports.

Open the Report Explorer using one of these approaches:

- From the MATLAB Toolstrip, in the **Apps** tab, in the **Database Connectivity and Reporting** section, click **Report Generator**.
- In the MATLAB Command Window, enter `report`.





The Report Explorer has three panes:

- The *Outline pane* on the left shows the hierarchy of components in currently opened report setup files. Report components can reside within other report components, creating parent, child, and sibling relationships.
- The *Library pane* in the middle lists the objects available in the context of the Outline pane.

Outline Pane Context	Library Pane Contents
No report setup file is open.	Reports
Report setup file is open.	Components
Style sheet is open.	Style sheet attributes

- The *Properties pane* contents depend on the Outline pane context. If no report setup file is open, on the right displays tasks the Report Explorer can perform. If a report setup file is open, the Properties pane displays the properties for the item that is currently selected in the Options pane.

Outline Pane Context	Properties Pane Contents
No report setup file is open.	Tasks that the Report Explorer can perform

<b>Outline Pane Context</b>	<b>Properties Pane Contents</b>
Report setup file is open.	Properties for the item that is currently selected  After you create a report setup file, the Properties pane initially displays properties for the report setup file as a whole.

---

**Tip** If the Report Explorer window opens with only two panes, one of the panes is hidden. You can move the vertical boundaries between the panes to reveal any hidden pane, or to make visible panes wider or narrower.

---

## **Interactive Report Generation Workflow**

- 1** Open the Report Explorer.
- 2** Create a report setup file. For details about report setups, see “Report Setup”.
- 3** Add content by adding to the report setup file existing components or custom components that you create. For details about using components, see “Work with Components” “Insert Components” on page 4-10.
- 4** Use Microsoft Word, HTML, or PDF templates to format reports. For details about templates, see “Report Templates” on page 7-2.
- 5** Generate the report. See “Generate Reports”.

## Acknowledgments

MATLAB Report Generator uses Antenna House® XSL Formatter. Antenna House is a trademark of Antenna House, Inc.

Antenna House XSL Formatter© 2009-2019 Copyright Antenna House, Inc.



# Create Your First Report

---

- “Create a Report Generator” on page 2-2
- “Maintain Interactive MATLAB Report” on page 2-10

## Create a Report Generator

This example shows how to create a simple report that explains and illustrates magic squares – matrices whose columns, rows, and diagonals each add up to the same number. See `magic`.

---

**Note** The complete example code is included after the step-by-step instructions.

---

- 1 Import the base classes.

To eliminate the need to use fully qualified names of `Report` and DOM API objects, use these statements. For example, instead of using `mlreportgen.report.Report`, you can use `Report`.

```
import mlreportgen.report.*
import mlreportgen.dom.*
```

- 2 Create a report object.

Create the report object. Use 'magic' as its file name and 'html' as its report type.

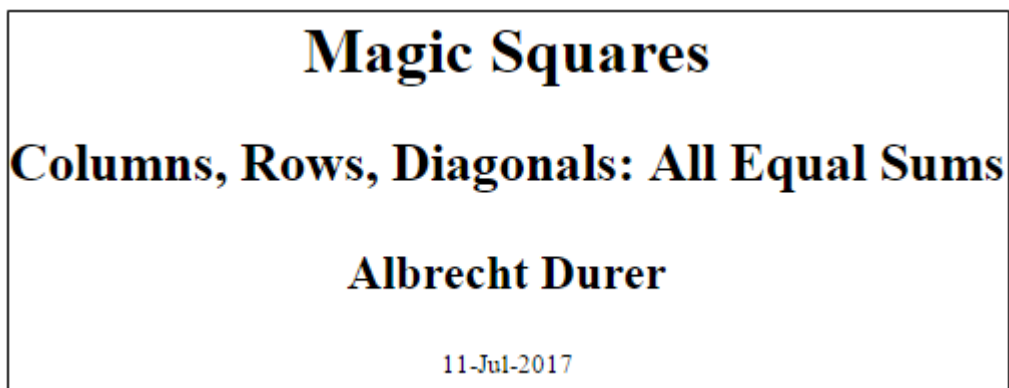
```
rpt = Report('magic','html');
```

To customize properties that apply to the whole report, see `mlreportgen.report.Report`.

- 3 Add a title page.

Create a title page and specify its title, subtitle and author. Then, add the title page to the report.

```
tp = TitlePage;
tp.Title = 'Magic Squares';
tp.Subtitle = 'Columns, Rows, Diagonals: All Equal Sums';
tp.Author = 'Albrecht Durer';
append(rpt,tp);
```



To customize additional title page properties, see `mlreportgen.report.TitlePage`.

- 4 Add a table of contents.

Add a default table of contents object to the report.

```
append(rpt,TableOfContents);
```

<input type="checkbox"/>	<b>Table of Contents</b>
<input checked="" type="checkbox"/>	<a href="#">Chapter 1. Introduction</a>
	<a href="#">Chapter 2. 10 x 10 Magic Square</a>
	<a href="#">Chapter 3. 25 x 25 Magic Square</a>

To customize the table of contents, see `mlreportgen.report.TableOfContents`.

## 5 Add a chapter and chapter sections.

Create a chapter object for the introduction and specify the chapter title. Add a section, add a paragraph to that section, and add that section to the chapter. Create another section and add a paragraph to it.

```
ch1 = Chapter;
ch1.Title = 'Introduction';
sec1 = Section;
sec1.Title = 'What is a Magic Square?';
para = Paragraph(['A magic square is an N-by-N matrix '...
'constructed from the integers 1 through N^2 '...
'with equal row, column, and diagonal sums.']);
append(sec1,para)
append(ch1,sec1)
sec2 = Section;
sec2.Title = 'Albrecht Durer and the Magic Square';
para = Paragraph([ ...
'The German artist Albrecht Durer (1471-1528) created '...
'many woodcuts and prints with religious and '...
'scientific symbolism. One of his most famous works, '...
'Melancholia I, explores the depressed state of mind '...
'which opposes inspiration and expression. '...
'Renaissance astrologers believed that the Jupiter '...
'magic square (shown in the upper right portion of '...
'the image) could aid in the cure of melancholy. The '...
'engraving's date (1514) can be found in the '...
'lower row of numbers in the square.']);
append(sec2,para)
append(ch1,sec2)
```

## Chapter 1. Introduction

### 1.1. What is a Magic Square?

A magic square is an  $N$ -by- $N$  matrix constructed from the integers 1 through  $N^2$  with equal row, column, and diagonal sums.

### 1.2. Albrecht Durer and the Magic Square

The German artist Albrecht Durer (1471-1528) created many woodcuts and prints with religious and scientific symbolism. One of his most famous works, *Melancholia I*, explores the depressed state of mind which opposes inspiration and expression. Renaissance astrologers believed that the Jupiter magic square (shown in the upper right portion of the image) could aid in the cure of melancholy. The engraving's date (1514) can be found in the lower row of numbers in the square.

For information on customizing chapters and sections, see `mlreportgen.report.Chapter` and `mlreportgen.report.Section` respectively.

#### 6 Add a figure.

Create an image of Durer in a figure window. Create the image in a MATLAB figure. Add the figure to the second section of introduction chapter and then, add the chapter to the report.

```
durerImage=load(which('durer.mat'),' -mat');
figure('Units','Pixels','Position',...
[200 200 size(durerImage.X,2)*.5 ...
size(durerImage.X,1)*.5 ]);
image(durerImage.X);
colormap(durerImage.map);
axis('image');
set(gca,'Xtick',[],'Ytick',[],...
'Units','normal','Position',[0 0 1 1]);
append(sec2,Figure)
append(rpt,ch1)
close(gcf)
```





For more information on figures, see `mlreportgen.report.Figure`. For more information on images, see `mlreportgen.report.FormalImage`.

## 7 Add a table.

Add another chapter object and specify its title. Specify the MATLAB code to create a 10-by-10 magic square. Add the results to a table and set these table properties:

- Row and column separators
- Table border
- Alignment of table entries

Then, add the table to the chapter and the chapter to the report.

```
ch2 = Chapter();  
ch2.Title = sprintf('10 x 10 Magic Square');
```

```
square = magic(10);  
tbl = Table(square);
```

```
tbl.Style = {...
    RowSep('solid','black','1px'),...
    ColSep('solid','black','1px'),};
tbl.Border = 'double';
tbl.TableEntriesStyle = {HAlign('center')};

append(ch2,tbl);
append(rpt,ch2);
```

## Chapter 2. 10 x 10 Magic Square

92	99	1	8	15	67	74	51	58	40
98	80	7	14	16	73	55	57	64	41
4	81	88	20	22	54	56	63	70	47
85	87	19	21	3	60	62	69	71	28
86	93	25	2	9	61	68	75	52	34
17	24	76	83	90	42	49	26	33	65
23	5	82	89	91	48	30	32	39	66
79	6	13	95	97	29	31	38	45	72
10	12	94	96	78	35	37	44	46	53
11	18	100	77	84	36	43	50	27	59

For more information on tables, see `mlreportgen.dom.Table`.

- 8 Add a MATLAB figure to a chapter.

Add another chapter object and specify its title. Specify the MATLAB code to create a 25-by-25 magic square and a color-coded figure of the magic square. Then, create a figure object and set its height, width, and caption. Add the figure to the chapter and the chapter to the report.

```
ch3 = Chapter();
ch3.Title = sprintf('25 x 25 Magic Square');

square = magic(25);
clf;
imagesc(square)
set(gca,'Ydir','normal')
axis equal
axis tight

fig = Figure(gcf);
fig.Snapshot.Height = '4in';
fig.Snapshot.Width = '6in';
fig.Snapshot.Caption = sprintf('25 x 25 Magic Square');

append(ch3,fig);
append(rpt,ch3);
delete(gcf)
```

## Chapter 3. 25 x 25 Magic Square

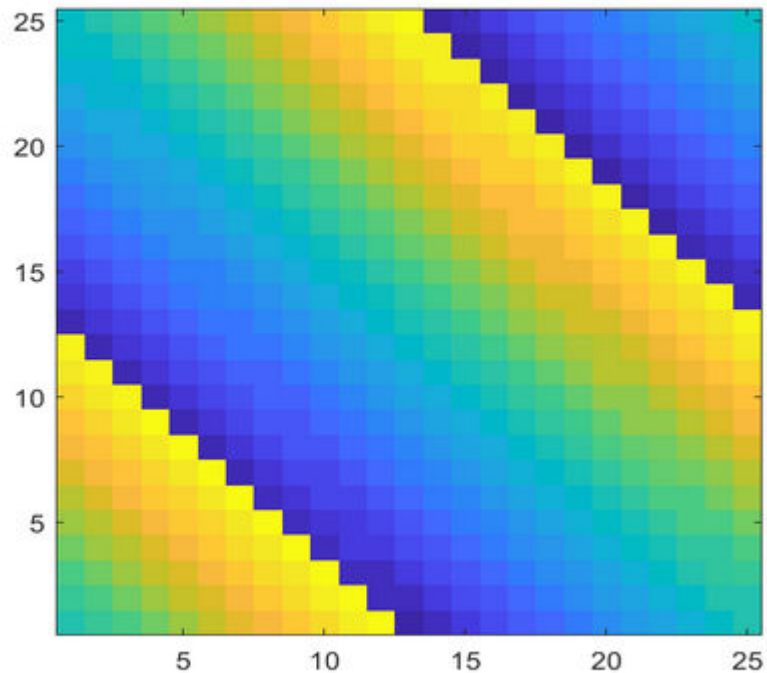


Figure 3.1. 25 x 25 Magic Square

For more information on figures, see `mlreportgen.report.Figure`.

**9** Close and run the report.

```
close(rpt)
rptview(rpt)
```

The complete code is:

```
import mlreportgen.report.*
import mlreportgen.dom.*
rpt = Report('magic','html');

tp = TitlePage;
tp.Title = 'Magic Squares';
tp.Subtitle = 'Columns, Rows, Diagonals: All Equal Sums';
tp.Author = 'Albrecht Durer';
append(rpt,tp);
append(rpt,TableOfContents);

ch1 = Chapter;
ch1.Title = 'Introduction';
sec1 = Section;
sec1.Title = 'What is a Magic Square?';
para = Paragraph(['A magic square is an N-by-N matrix '...
'constructed from the integers 1 through N^2 '...
'with equal row, column, and diagonal sums.']);
append(sec1,para)
append(ch1,sec1)

sec2=Section;
sec2.Title = 'Albrecht Durer and the Magic Square';
para = Paragraph([ ...
'The German artist Albrecht Durer (1471-1528) created '...
'many woodcuts and prints with religious and '...
'scientific symbolism. One of his most famous works, '...
'Melancholia I, explores the depressed state of mind '...
'which opposes inspiration and expression. '...
'Renaissance astrologers believed that the Jupiter '...
'magic square (shown in the upper right portion of '...
'the image) could aid in the cure of melancholy. The '...
'engraving's date (1514) can be found in the '...
'lower row of numbers in the square.']);
append(sec2,para)
append(ch1,sec2)

durerImage=load(which('durer.mat'),' -mat');
figure('Units','Pixels','Position',...
[200 200 size(durerImage.X,2)*.5 ...
size(durerImage.X,1)*.5 ]);
image(durerImage.X);
colormap(durerImage.map);
axis('image');
set(gca,'Xtick',[],'Ytick',[],...
'Units','normal','Position',[0 0 1 1]);
append(sec2,Figure)
append(rpt,ch1)
close(gcf)
```

```
ch2 = Chapter();
ch2.Title = sprintf('10 x 10 Magic Square');
square = magic(10);
tbl = Table(square);
tbl.Style = {...
RowSep('solid','black','1px'),...
ColSep('solid','black','1px'),};
tbl.Border = 'double';
tbl.TableEntriesStyle = {HAlign('center')};
append(ch2,tbl);
append(rpt,ch2);

ch3 = Chapter();
ch3.Title = sprintf('25 x 25 Magic Square');
square = magic(25);
clf;
imagesc(square)
set(gca,'Ydir','normal')
axis equal
axis tight
fig = Figure(gcf);
fig.Snapshot.Height = '4in';
fig.Snapshot.Width = '6in';
fig.Snapshot.Caption = sprintf('25 x 25 Magic Square');
append(ch3,fig);
append(rpt,ch3);

delete(gcf)
close(rpt)
rptview(rpt)
```

## See Also

rptview

## Maintain Interactive MATLAB Report

This example shows how a basic report was created using the Report Explorer. This report explains and illustrates magic squares – matrices whose columns, rows, and diagonals each add up to the same number (see the `magic` function reference in the MATLAB documentation).

To create this report, you perform these main tasks:

- “Create a Report Setup File” on page 2-10
- “Add Report Content Using Components” on page 2-11

---

**Note** You do not need to know the MATLAB software to use this example. However, knowledge of MATLAB is helpful for understanding the MATLAB code that executes during report generation.

---

This example includes separate sections for different kinds of report creation and generation tasks. Each section builds on the previous sections. However, if you want to work through a later section without having done the previous sections, you can view the completed report setup file: `Magic Squares Report`.

### Create a Report Setup File

To set up the magic squares report, first create a setup file to store the setup. Then add MATLAB objects, called components, to the setup to specify the report content.

To create the report setup file:

- 1 Start a MATLAB software session.
- 2 Open the Report Explorer. From the MATLAB Toolstrip, in the **Apps** tab, in the **Database Connectivity and Reporting** section, click **Report Generator**.
- 3 Select **File > New** to create a report setup file. The new report setup has the default name `Unnamed.rpt`.
- 4 In the Properties pane on the right:
  - a To save the report in the current working folder, select **Present working directory** from the **Directory** list.
  - b Set **File format** to `HTML (from template)` to generate the report output as HTML. Using the `(from template)` option creates the report table of contents in a format that you can expand and collapse.
  - c In the **Report description** text box, replace the existing text with the following text.

This report creates a series of magic squares and displays them as images.

A magic square is a matrix in which the columns, rows, and diagonal all add up to the same number.

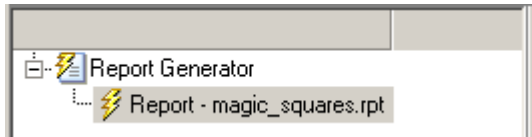
---

**Note** When you change a Properties pane field, its background color changes. This indicates that there are unapplied changes to that field. As soon as you perform any action with another

component, MATLAB Report Generator applies the changes, and the background color becomes white again.

- 5 Save your report. Select **File > Save As** and name your report setup file `magic_squares.rpt`.

The new file name appears in the Outline pane.



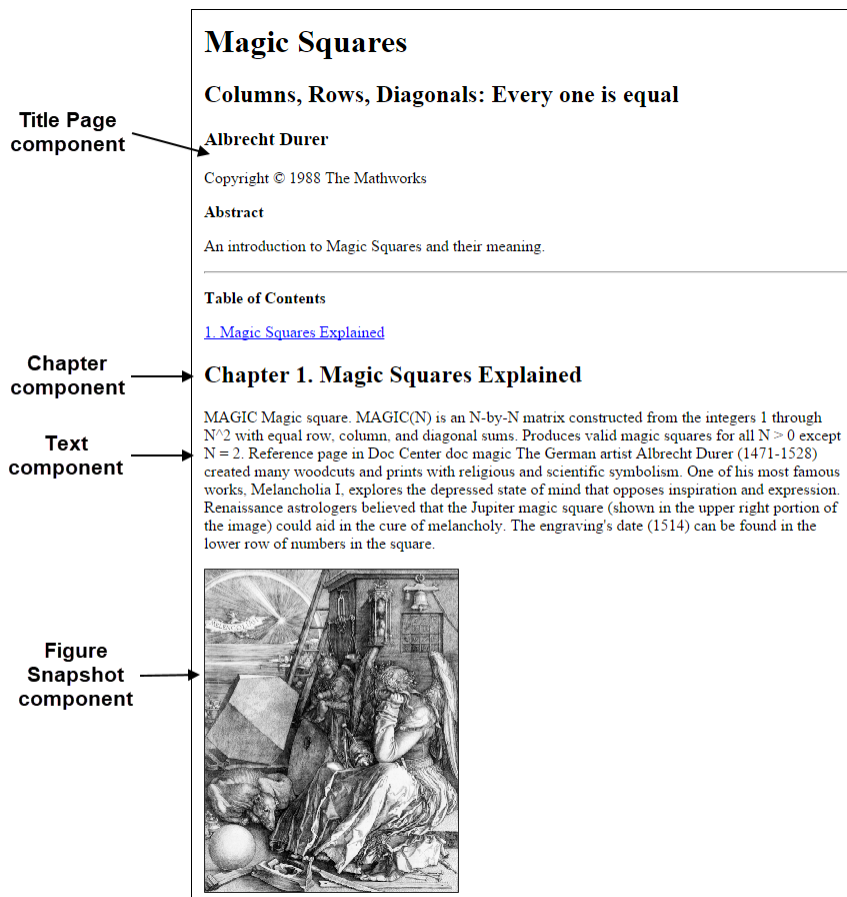
To create the content for the report, see “Add Report Content Using Components” on page 2-11.

## Add Report Content Using Components

- “Report Components” on page 2-11
- “Specify Report Variables” on page 2-12
- “Create a Title Page” on page 2-14
- “Add a Chapter” on page 2-16
- “Add Introductory Text to the First Chapter” on page 2-17
- “Add an Image” on page 2-18
- “Create the Magic Squares and Their Images” on page 2-21
- “Create a For Loop” on page 2-21
- “Add a Chapter for Each Square” on page 2-23
- “Determine the Matrix Size” on page 2-23
- “Insert the Magic Square Size into the Report” on page 2-24
- “Create the Magic Square” on page 2-25
- “Add Display Logic” on page 2-26
- “Display the Magic Square” on page 2-29

### Report Components

Report components specify the information to include in the report. The following figure shows a sample page from the report that you create in this example, highlighting components that you use to produce the report.



## Specify Report Variables

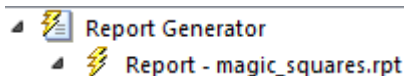
The magic squares report uses variables defined in the MATLAB workspace to specify the number and sizes of squares to display and whether to display the variables as tables of numbers or images of color-coded squares:

- The *magicSizeVector* variable specifies an array of magic square sizes
- *largestDisplayedArray* variable specifies the size of the largest magic square to be displayed as an array of numbers

You could require that a user create these variables in the MATLAB workspace before running the report. However, a better solution is to let the report create the variables, using the Evaluate MATLAB Expression component.

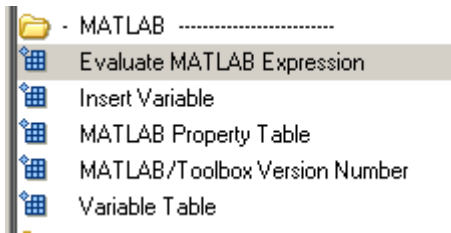
To use the **Evaluate MATLAB Expression** component to define the report variables:

- 1 In the Outline pane on the left, select the root component of the report setup.



- 2 In the Library pane in the middle, under **MATLAB**, select **Evaluate MATLAB Expression**.

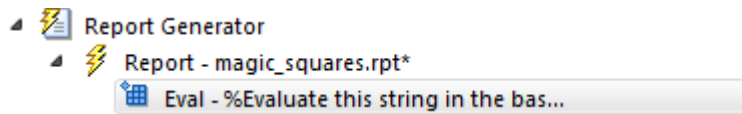




- 3 In the Properties pane on the right, click the icon next to **Add component to current report** to insert the **Evaluate MATLAB Expression** component into the report.

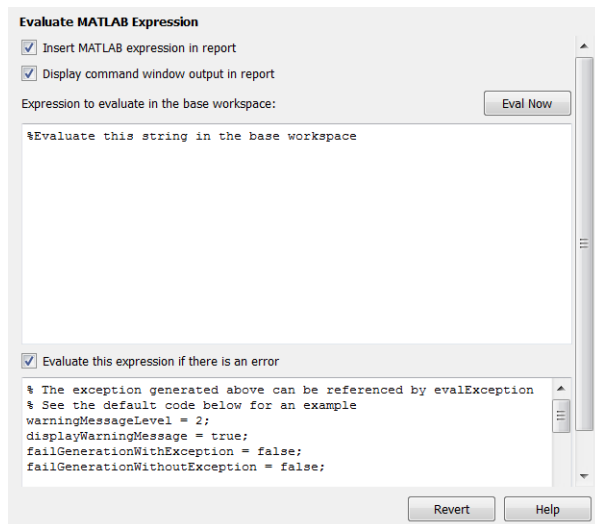
You can edit the component information in the Properties pane only after you add the component to the report.

In the Outline pane, the Eval component appears under the magic\_squares report.



The icon in the upper left corner of the Eval component indicates that this component cannot have child components. By default, any components you add with the Eval component selected are siblings to this component.

The options for the **Evaluate MATLAB Expression** component appear in the Properties pane.



- 4 To exclude the MATLAB code details and its output in this report, clear the **Insert MATLAB expression in report** and **Display command window output in report** check boxes.
- 5 In the **Expression to evaluate in the base workspace** text box, replace the existing text with the following MATLAB code.

---

**Tip** Copy and paste this text from the HTML documentation into the Report Explorer.

---

```
%This MATLAB code sets up two variables
%that define how the report runs.
```

```
%magicSizeVector is a list of MxM  
%Magic Square sizes to insert into  
%the report. Note that magic  
%squares cannot be 2x2.
```

```
magicSizeVector=[4 8 16 32];
```

```
%largestDisplayedArray sets the  
%limit of array size that will be  
%inserted into the report with the  
%Insert Variable component.
```

```
largestDisplayedArray=15;
```

- 6 In the **Evaluate this expression if there is an error** text box, replace the existing text with the following text.

```
disp(['Error during eval: ', evalException.message])
```

This causes an error to display if the MATLAB code fails.

---

**Tip** To execute these commands immediately, in the top right corner of the Report Explorer, click the **Eval Now** button. This confirms that your commands are correct, to reduce the chances of report generation problems.

---

- 7 Save the report..

### Create a Title Page

---

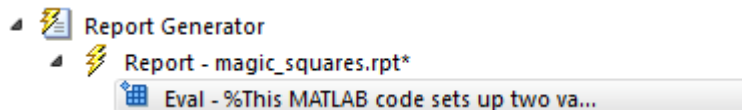
**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Maintain Interactive MATLAB Report” on page 2-10.

If you have not completed the previous sections of this example, open the completed report setup file: **Magic Squares Report**.

---

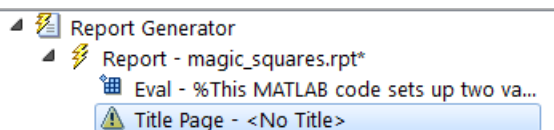
To create a title page for the report, use the **Title Page** component.

- 1 In the Outline pane on the left, select the Eval component.



- 2 In the Options pane in the middle, under **Formatting**, add the **Title Page** component to the report.

Because the Eval component icon indicates that this component cannot have children, the **Title Page** component is a sibling of the Eval component. Likewise, the **Title Page** component cannot have child components.

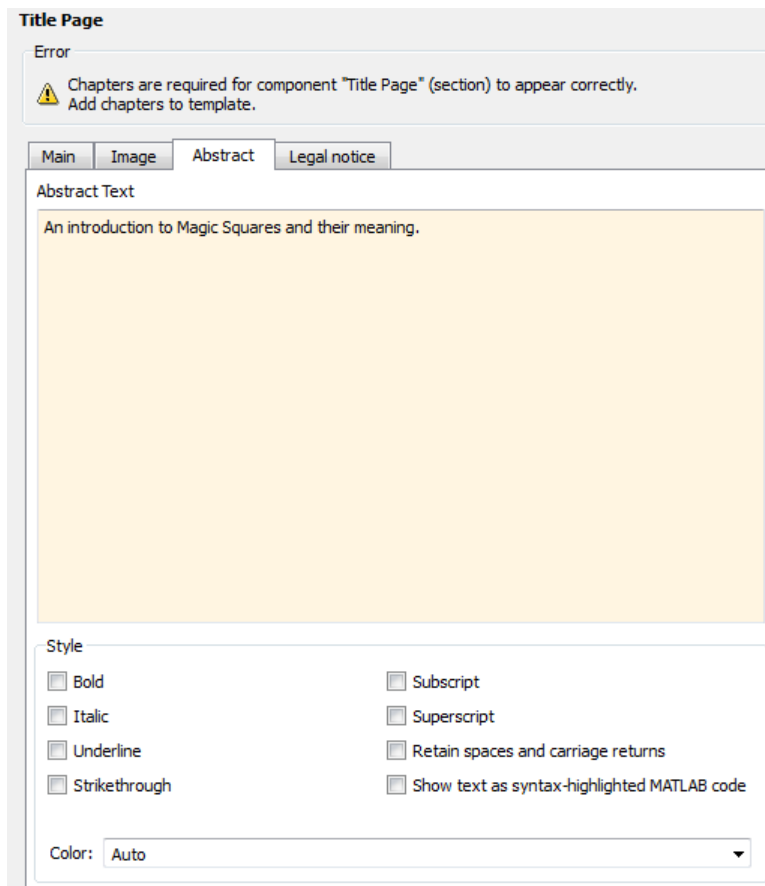


**Note** To use a **Title Page** component, your report needs a **Chapter** component. You have not yet added a **Chapter** component, so the Properties pane displays a message indicating that a chapter is required for the **Title Page** component to appear correctly. Because later in this example you add **Chapter** components to this report, you can ignore that message.

- 3 In the Properties pane on the right, use the **Main** tab to enter the title page information.
  - a In the **Title** text box, enter Magic Squares.
  - b In the **Subtitle** text box, enter Columns, Rows, Diagonals: Everyone is Equal.
  - c Under **Options**, choose Custom author from the list.

- d In the field to the right of the **Custom author** field, enter Albrecht Durer.  
 Albrecht Dürer created an etching that contains a magic square. Your final report includes an image of that etching.
- e Select the **Include copyright holder and year** check box.
- f In the next text box, enter The MathWorks.
- g In the second text box, enter 1988.

- 4 In the Properties pane, in the **Abstract** tab, enter:  
 An introduction to Magic Squares and their meaning.



- 5 Save the report.

### Add a Chapter

---

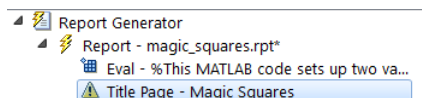
**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Maintain Interactive MATLAB Report” on page 2-10.

If you have not completed the previous sections of this example, open the completed report setup file: **Magic Squares Report**.

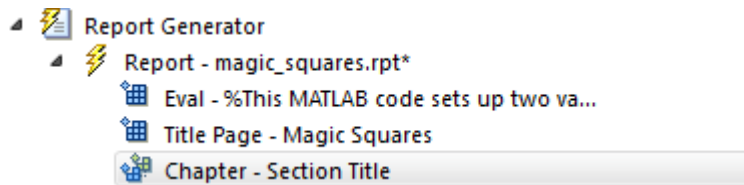
---

Add a chapter to the report by using the **Chapter/Subsection** component.

- 1 In the Outline pane on the left, select the **Title Page** component.



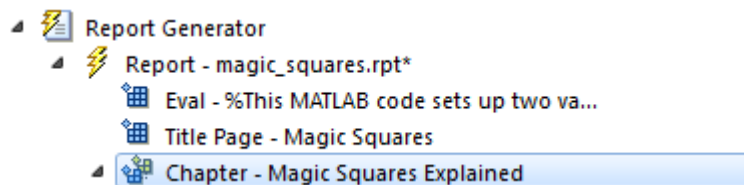
- 2 In the Library pane in the middle, under **Formatting**, add a **Chapter/Subsection** component.



The Eval, **Title Page**, and **Chapter** components are all child components of the report's top level and are siblings of one another.

- For the custom chapter title, in the Properties pane on the right, enter **Magic Squares Explained**.

The Outline pane displays the chapter title.



- Save the report.

### Add Introductory Text to the First Chapter

---

**Note** This section builds on the previous tasks described in the step-by-step example summarized in "Maintain Interactive MATLAB Report" on page 2-10.

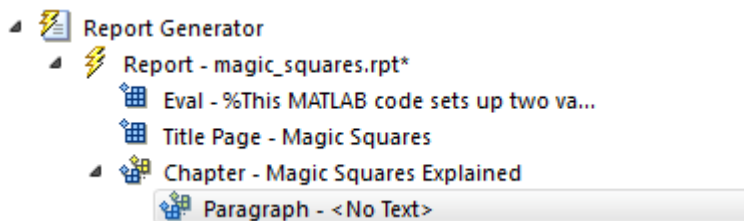
If you have not completed the previous sections of this example, open the completed report setup file: **Magic Squares Report**.

---

Include introductory text in the first chapter by adding the **Paragraph** and **Text** components.

- In the Outline pane on the left, select the **Chapter** component.
- In the Library pane in the middle, under **Formatting**, add a **Paragraph** component.

In the Outline pane, the new component appears as a child of the **Chapter** component.



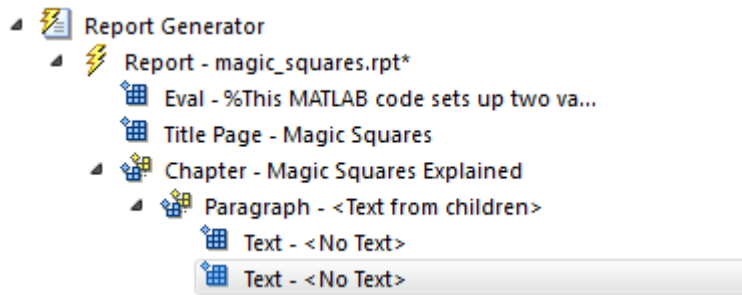
- By default, the **Paragraph** component inherits its text from its child components. Add two **Text** components.

---

**Note** The **Text** component must have a **Paragraph** component as its parent.

---

- In the Library pane, under the **Formatting** category, add two **Text** components to the outline.



- 5 In the Outline pane, select the first Text component.
- 6 In the **Text to include in report** text box, enter `%<help('magic')>`.

The % sign and angle brackets <> indicate to the MATLAB Report Generator software that this is MATLAB code to evaluate. The command `help('magic')` displays information about the MATLAB `magic` function.

- 7 In the Outline pane, select the second Text component.
- 8 In the **Text to include in report** text box, enter this text.

```
The German artist Albrecht Durer (1471-1528)
created many woodcuts and prints with religious
and scientific symbolism. One of his most famous
works, Melancholia I, explores the depressed state
of mind that opposes inspiration and expression.
Renaissance astrologers believed that the Jupiter magic
square (shown in the upper right portion of the image)
could aid in the cure of melancholy. The engraving's
date (1514) can be found in the lower row of numbers
in the square.
```

- 9 Save the report.

The contents of the first chapter are now complete.

### Add an Image

---

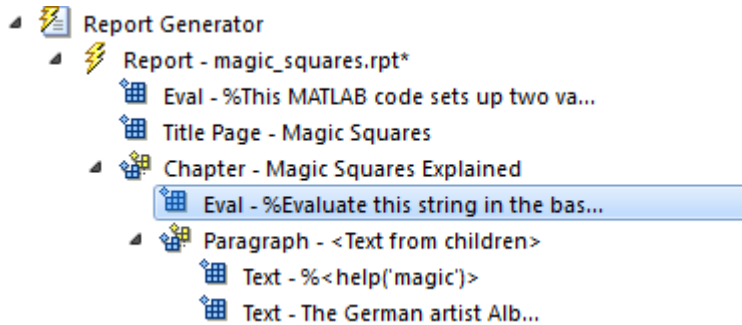
**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Maintain Interactive MATLAB Report” on page 2-10.

If you have not completed the previous sections of this example, open the completed report setup file: Magic Squares Report.

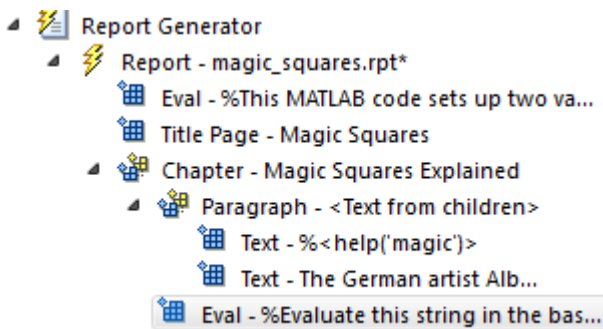
---

Create an image of Albrecht Dürer and include it in the report.

- 1 In the Outline pane on the left, select the **Chapter** component.
- 2 In the Library pane in the middle, under **MATLAB**, add an **Evaluate MATLAB Expression** component.



- 3 Move the **Eval** component under the **Paragraph** component so that the image follows the introductory text. To move it, on the toolbar, click the **down** arrow.



- 4 With the Eval component selected, set these properties:

- a Clear the **Insert MATLAB expression in report** and **Display command window output in report** check boxes. You do not want to include the code or its output in the report.
- b In the **Expression to evaluate in the base workspace** text box, replace the existing text with this MATLAB code.

```
%This loads a self-portrait of Albrecht
%Durer, a German artist. There is a
%magic square in the upper right corner
%of the image.

durerData=load('durer.mat','-mat');
figure('Units','Pixels',...
'Position',[200 200 size(durerData.X,2)*.5 size(durerData.X,1)*.5 ]);

image(durerData.X);
colormap(durerData.map);
axis('image');
set(gca,...
'Xtick',[],...
'Ytick',[],...
'Units','normal',...
'Position',[0 0 1 1]);

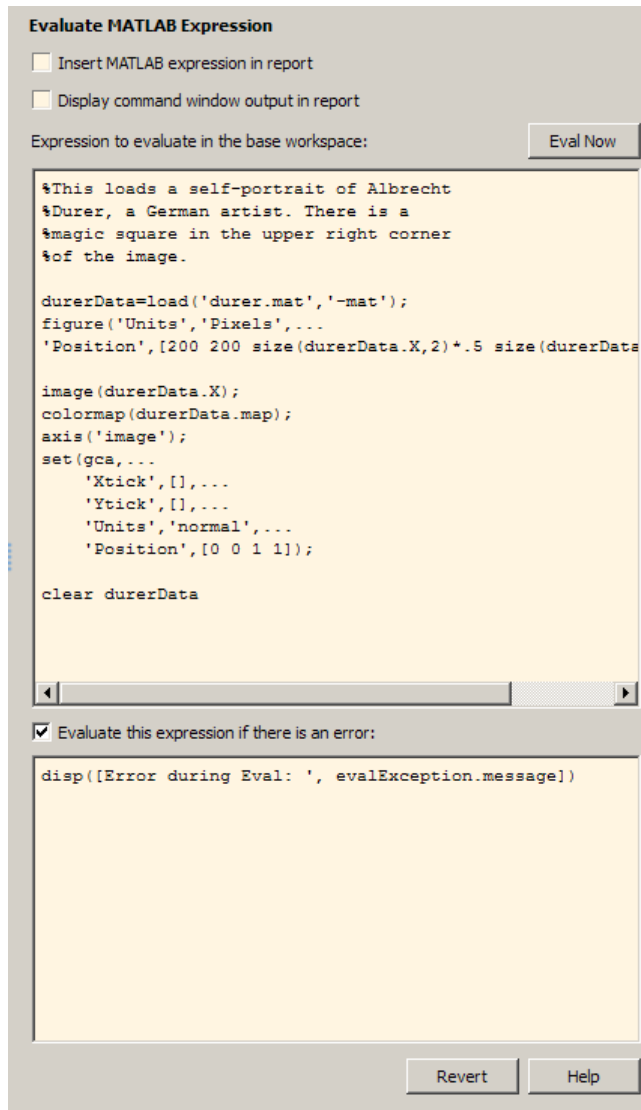
clear durerData
```

This MATLAB code displays the Dürer etching in a MATLAB figure window.

- c In the **Evaluate this expression if there is an error** text box, replace the existing text with the following text:

```
disp(['Error during eval: ', evalException.message])
```

This code executes if an error occurs while loading the Dürer etching.



- 5 In the Outline pane on the left, select the Eval component.
- 6 In the Library pane in the middle, under **Handle Graphics**, add a **Figure Snapshot** component to the report.

To inline an image component (such as **Image** or **Figure Snapshot**), include it in a **Paragraph** component.

- 7 In the Properties pane:
  - a In the **Paper** orientation list, select Portrait.
  - b In the **Invert hardcopy** list, select Don't invert.

Selecting this option specifies not to change the image's onscreen colors for printing.

The next three steps set up the report to delete the image from the MATLAB workspace after the image has been added to the report.

- 8 In the Outline pane, select the **Figure Snapshot** component.



- 9 In the Library pane, under **MATLAB**, add an **Evaluate MATLAB Expression** component to the report.
- 10 In the Properties pane:
  - a Clear the **Insert MATLAB expression in report** and **Display command window output in report** check boxes. You do not want to include the code or its output in the report.
  - b In the **Expression to evaluate in the base workspace** text box, replace the existing text with:
 

```
%This command deletes the Durer image
delete(gcf);
```

The `delete(gcf)` command deletes the current image in the MATLAB workspace, in this case, the Dürer etching.
  - c In the **Evaluate this expression if there is an error** text box, replace the existing text with the following text:
 

```
disp(['Error during eval: ', evalException.message])
```

This code executes if an error occurs while deleting the Dürer etching.
- 11 Save the report.

### Create the Magic Squares and Their Images

Add a chapter to the report for each magic square specified by the *magicSizeVector* report variable. You use a **For Loop** component to perform this repetitive task. To create the magic squares and their images, you perform these tasks:

- “Create a For Loop” on page 2-21
- “Add a Chapter for Each Square” on page 2-23
- “Determine the Matrix Size” on page 2-23
- “Insert the Magic Square Size into the Report” on page 2-24
- “Create the Magic Square” on page 2-25
- “Add Display Logic” on page 2-26
- “Display the Magic Square” on page 2-29

### Create a For Loop

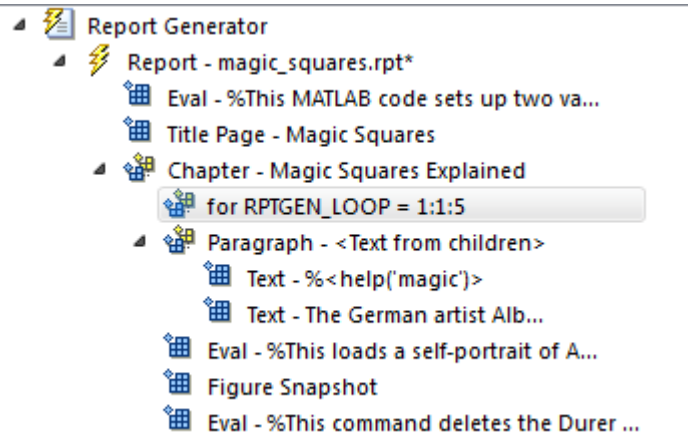
---

**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Maintain Interactive MATLAB Report” on page 2-10.

If you have not completed the previous sections of this example, open the completed report setup file: **Magic Squares Report**.

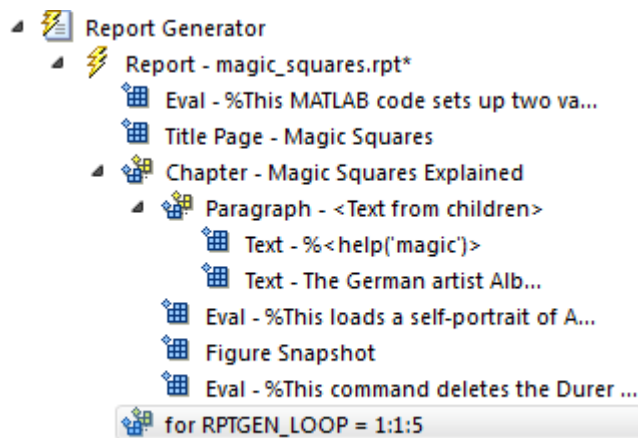
---

- 1 In the Outline pane on the left, select the **Chapter** component.
- 2 In the Library pane in the middle, under **Logical and Flow Control**, add a **For Loop** component to the report.



This **For Loop** component appears inside the **Chapter** component. However, the magic squares should be processed after the first chapter, so make the `for` component a sibling of the **Chapter** component, not a child.

- 3 In the Outline pane, select the `for` component.
- 4 Click the **left** arrow to make the `for` component a sibling of the **Chapter** component.



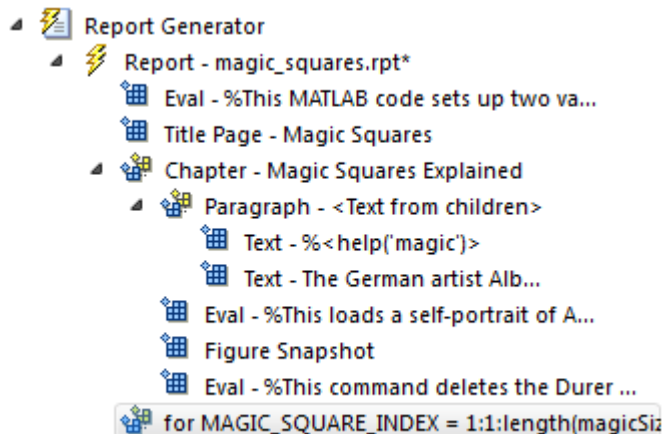
- 5 In the Properties pane on the right:
  - a In the **End** text box, replace the existing text with this text:
 

```
length(magicSizeVector)
```

This is the length of the vector that contains the various sizes for the magic square matrices.
  - b In the **Variable name** text box, replace the existing text with the following text:
 

```
MAGIC_SQUARE_INDEX
```

This variable acts as a loop index.



- 6 Save the report.

### Add a Chapter for Each Square

---

**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Maintain Interactive MATLAB Report” on page 2-10.

If you have not completed the previous sections of this example, open the completed report setup file: **Magic Squares Report**.

---

Create a chapter for each square by adding a **Chapter** component to the report as a child of the **For Loop** component. This causes the Report Generator to create a chapter on each iteration of the for loop during report generation.

- 1 In the Outline pane on the left, select the **for** component.
- 2 In the Library pane in the middle, under **Formatting**, add a **Chapter/Subsection** component to the report setup.

It becomes a child of the **for** component.

- 3 In the Properties pane on the right, select **Custom** from the **Title** list and enter this for the chapter title:

Magic Square # %<MAGIC\_SQUARE\_INDEX>

- 4 Save the report.

### Determine the Matrix Size

---

**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Maintain Interactive MATLAB Report” on page 2-10.

If you have not completed the previous sections of this example, open the completed report setup file: **Magic Squares Report**.

---

Extract the size of each magic square matrix from `magicSizeVector` using an **Evaluate MATLAB Expression** component.

- 1 In the Outline pane on the left, select the bottom **Chapter** component.
- 2 In the Library pane in the middle, under **MATLAB** category, add an **Evaluate MATLAB Expression** component to the report setup.
- 3 In the Properties pane:
  - a Clear the **Insert MATLAB expression in report** and **Display command window output in report** check boxes.
  - b In the **Expression to evaluate in the base workspace** text box, replace the existing text with:

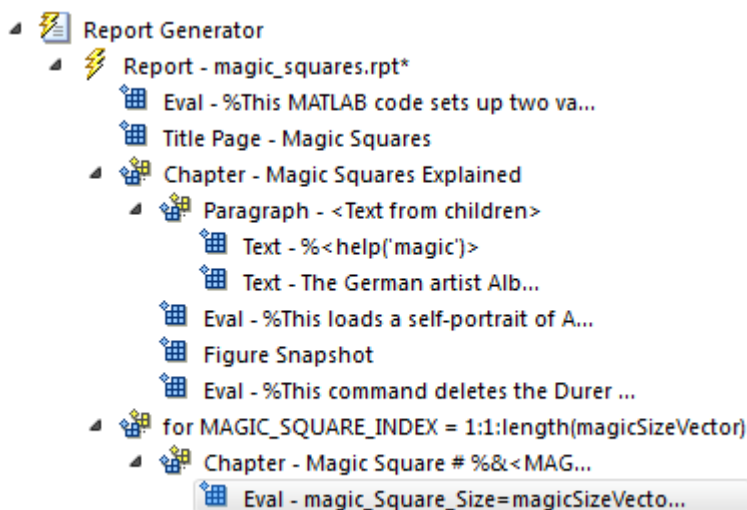
```
magic_Square_Size=magicSizeVector(MAGIC_SQUARE_INDEX);
```

This command extracts the next size for the magic square from the vector of sizes initialized in the first Eval component of the report. The variable `magic_Square_Size` represents the size of the current magic square being processed.

- c In the **Evaluate this expression if there is an error** text box, replace the existing text with this text:

```
disp(['Error during eval: ', evalException.message])
```

This code executes if an error occurs while attempting to extract a value from `magicSizeVector`.



- 4 Save the report.

### Insert the Magic Square Size into the Report

---

**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Maintain Interactive MATLAB Report” on page 2-10.

If you have not completed the previous sections of this example, open the completed report setup file: **Magic Squares Report**.

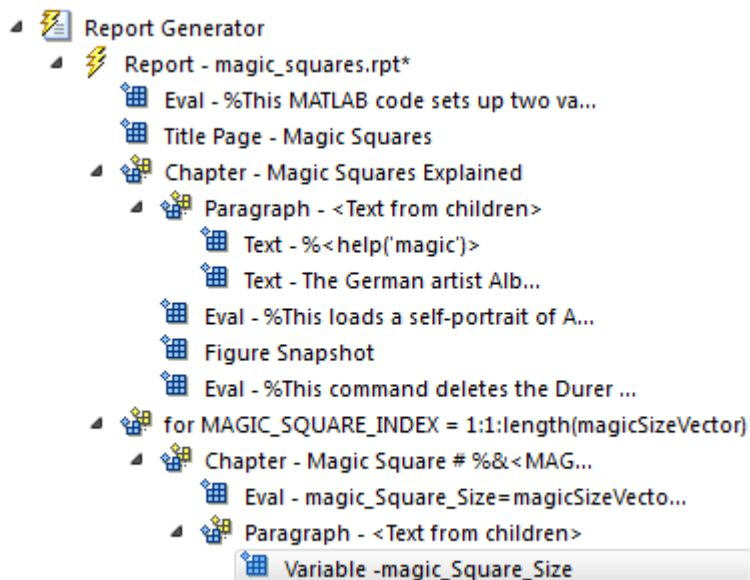
---

Insert the size of the magic square into the report using the **Paragraph** and **Insert Variable** components.

- 1 In the Outline pane on the left, select the bottom **Eval** component.
- 2 In the Library pane in the middle, under **Formatting**, add a **Paragraph** component to the report setup.

Do not change the properties. The variable that contains the size of the magic square goes in this paragraph.

- 3 In the Outline pane, select the **Paragraph** component (below the for component).
- 4 In the Library pane, under **MATLAB**, add an **Insert Variable** component into the report setup.
- 5 In the Properties pane on the right:
  - a In the **Variable name** text box, enter `magic_Square_Size`.
  - b From the **Display as** list, select **Inline text**.



- 6 Save the report.

### Create the Magic Square

**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Maintain Interactive MATLAB Report” on page 2-10.

If you have not completed the previous sections of this example, open the completed report setup file: **Magic Squares Report**.

To create the magic square and display the associated matrix or image, use the **Evaluate MATLAB Expression** component.

- 1 In the Outline pane on the left, select the bottom **Paragraph** component.

- 2 In the Library pane in the middle, under **MATLAB**, add an **Evaluate MATLAB Expression** component to the report setup.
- 3 Make this **Evaluate MATLAB Expression** component a sibling of the **Paragraph** component. In the Outline pane, select the **Eval** component. Click the left arrow on the toolbar.
- 4 In the Properties pane on the right:

- a Clear the **Insert MATLAB expression in report** and **Display command window output in report** check boxes.
- b In the **Expression to evaluate in the base workspace** text box, replace the existing text with this MATLAB code.

```
% This MATLAB script produces a magic
% square of size magic_Square_Size
% and creates an image of that square.

mySquare=magic(magic_Square_Size);
clf
imagesc(mySquare);
title(sprintf('Magic Square N=%i',magic_Square_Size))
set(gca,'Ydir','normal');
axis equal;
axis tight;
```

This code creates a magic square matrix `mySquare` of size `magic_Square_Size`, and opens an image of that matrix in the MATLAB figure window.

- c In the **Evaluate this expression if there is an error** text box, replace the existing text with this text:

```
disp(['Error during eval: ', evalException.message])
```

This code executes if an error occurs while creating and displaying the magic square.

- 5 Save the report.

### Add Display Logic

---

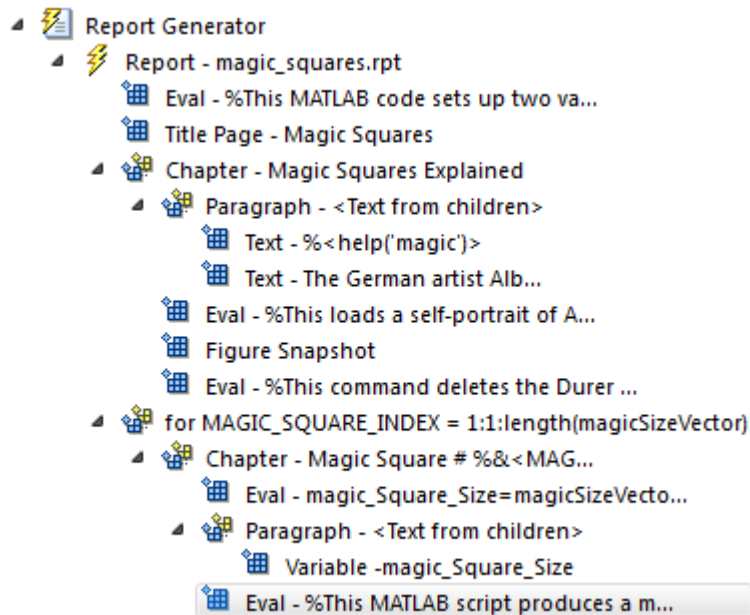
**Note** This section builds on the previous tasks described in the step-by-step example summarized in “Maintain Interactive MATLAB Report” on page 2-10.

If you have not completed the previous sections of this example, open the completed report setup file: **Magic Squares Report**.

---

Use `Logical If`, `Logical Then`, and `Logical Else` components to determine whether to display the magic square as an array of numbers or as an image.

- 1 In the Outline pane on the left, select the `Eval` component.



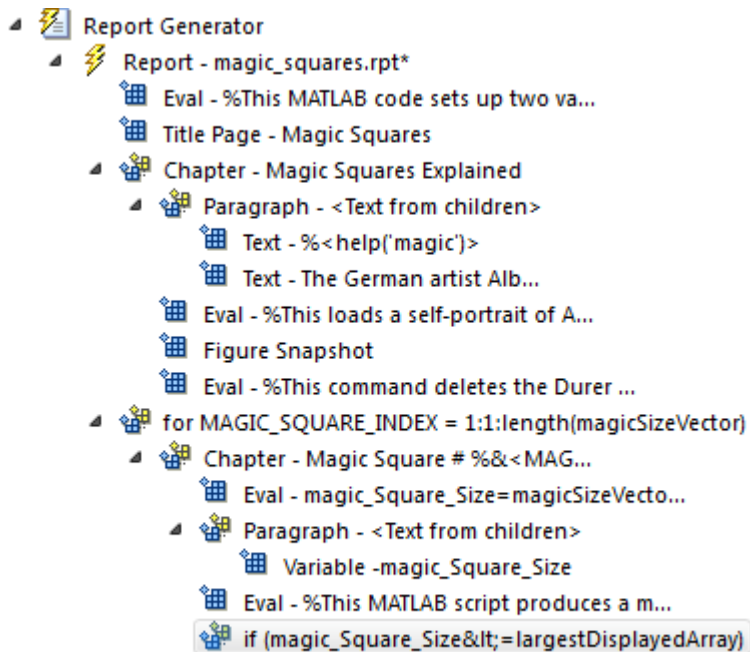
- 2 On the Library pane in the middle, under **Logical and Flow Control**, double-click **Logical If**.
- 3 On the Properties pane on the right, in the **Test Expression** text box, replace the existing text with this text:

```
magic_Square_Size<=largestDisplayedArray
```

This command tests whether the current matrix size (`magic_Square_Size`) is less than or equal to the value assigned in the first `Eval` component of the report (`largestDisplayedArray=15`).

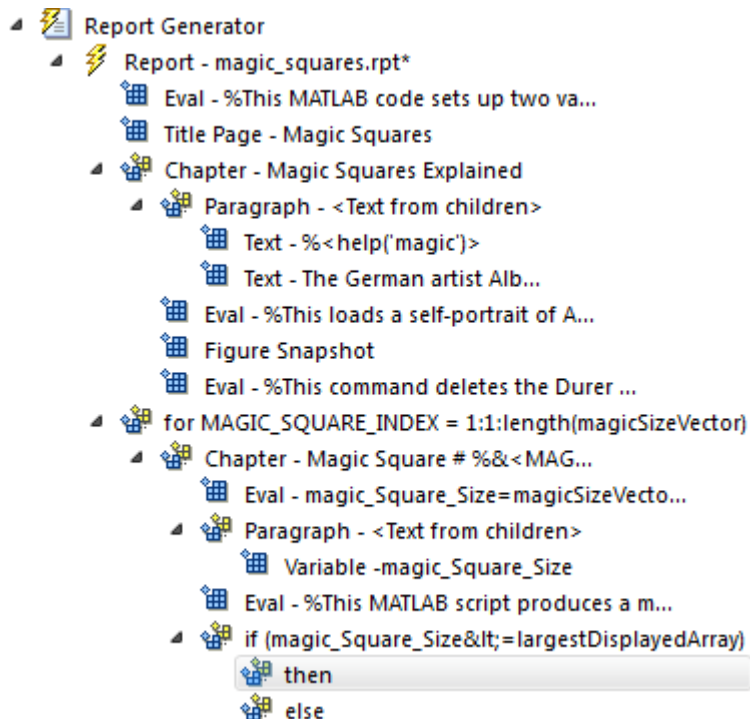
To process the result of this **Logical If** component, create two child components—**Logical Then** and **Logical Else**. If `magic_Square_Size` is less than or equal to 15, the matrix variable appears in the report. If `magic_Square_Size` is greater than 15, the matrix image appears in the report.

- 4 On the Outline pane, select the `if` component.



- 5 On the Library pane, under **Logical and Flow Control**, double-click **Logical Else**.
- 6 On the Outline pane, select the `if` component again.
- 7 On the Library pane, under **Logical and Flow Control**, double-click **Logical Then**.

The `then` component appears above the `else` component.





- 8 Save the report.

### Display the Magic Square

---

**Note** This section builds on the step-by-step example presented in “Maintain Interactive MATLAB Report” on page 2-10.

To see the completed report setup file, open **Magic Squares Report**.

---

- 1 In the Outline pane on the left, select the **then** component.
- 2 In the Library pane in the middle, under **MATLAB**, double-click **Insert Variable**.
- 3 In the Properties pane on the right:
  - a In the **Variable name** text box, enter `mySquare`, which is the variable that contains the magic square of the specified size.
  - b From the **Title** list, select **None**.
  - c In the **Array size limit** text box, enter `0`.

This **Variable** component displays the magic square matrix, stored in the `mySquare` variable.

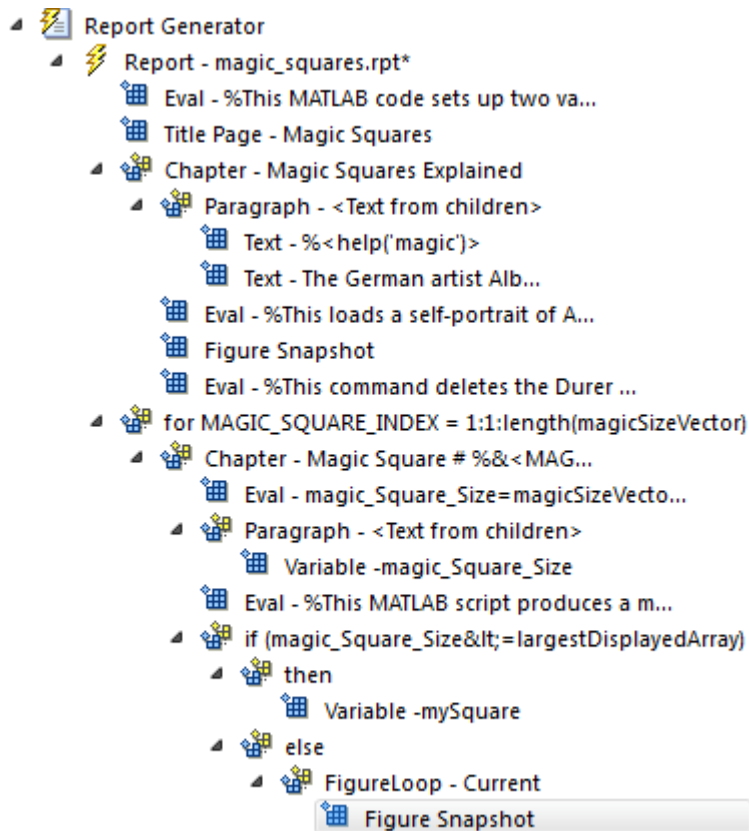
- 4 In the Outline pane, select the **else** component.
- 5 In the Library pane, under **Handle Graphics**, double-click **Figure Loop**.

Do not change its properties.

- 6 In the Outline pane, select the **Figure Loop** component.
- 7 In the Library pane, under **Handle Graphics**, double-click **Figure Snapshot**.
- 8 In the Properties pane:
  - a In the **Paper orientation** list, select **Portrait**.
  - b In the **Image size** list, select **Custom**.
  - c Under the **Image size** list, enter `[5 4]` for the custom image size.
  - d In the **Invert hardcopy** list, select **Invert**.

This option changes dark axes colors to light axes colors, and vice versa.

The Outline pane looks like this.



9 Save the report.

## Error Handling for MATLAB Code

You can add MATLAB code to a report, by using the **Evaluate MATLAB Expression** component (also called the Eval component).

The **Evaluate MATLAB Expression** component dialog box includes an **Evaluate this expression if there is an error** check box. The dialog box includes default error handling code that you can use, or you can create your own error handling code.

If you do not change the default error handling code, then when you generate the report, and there is an error in the MATLAB code that you added:

- If you clear **Evaluate this expression if there is an error** check box, then the complete report is generated, without displaying an error message at the MATLAB command line.
- If you select **Evaluate this expression if there is an error** check box, then the complete report is generated and an error message appears at the MATLAB command line.

To stop report generation when an error occurs in the MATLAB code that you added, change the second and third lines of the following default error handling code, as described below:

```
warningMessageLevel = 2;
displayWarningMessage = true;
```

```
failGenerationWithException = false;
failGenerationWithoutException = false;
```

To stop report generation and display an exception, change the default code to:

```
displayWarningMessage = false;
failGenerationWithException = true;
```

To stop report generation without displaying an exception, change the default code to:

```
displayWarningMessage = false;
failGenerationWithoutException = true;
```

If you want to replace the default error handling code, use the `evalException.message` variable in your code to return information for the exception.

## Generate a Report

---

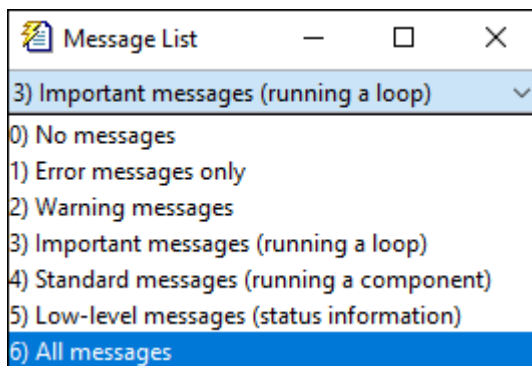
**Note** This section builds on the step-by-step example presented in “Maintain Interactive MATLAB Report” on page 2-10.

To see the completed report setup file, open [Magic Squares Report](#).

---

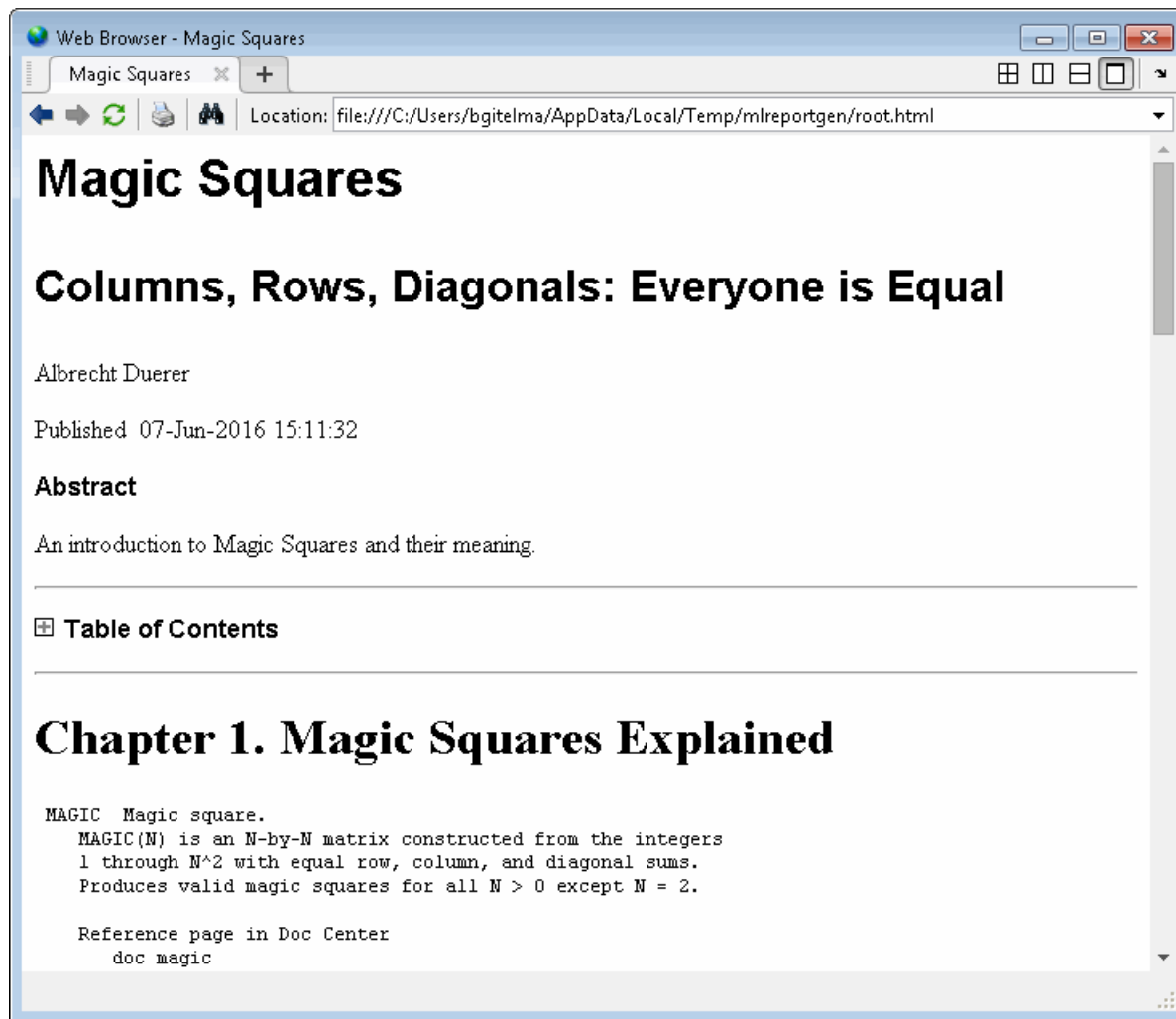
At the beginning of this example, you specified HTML (from `template`) as the output format of the report. If you did not, or if you opened the completed report setup file later in the tutorial, set it that way now.

- 1 On the toolbar, click the **Report** button to generate the report.
- 2 While the report generates, in the Message List window, specify the level of detail you want the Message List window to display.



In the Report Explorer Outline pane, each component of the report setup file highlights as it executes.

When processing finishes, the MATLAB Web browser opens and displays the HTML file.



When you choose HTML (from `template`) as the output type, the table of contents appears collapsed in the report. Expand or collapse each node by clicking the plus or minus sign on the node. Press **Ctrl**+click to expand or collapse the entire structure (**CMD**+click on Macintosh platforms).

# Report API Reports

---

## Open a Template File

Each report output type has a default template file for each output type, Microsoft Word, PDF, and HTML. Each template file contains a template library of document part templates. The names of document part templates for a reporter start with the name of that reporter to distinguish them from built-in Word templates. For Word output, the template library is in a Word template file named `default.dotx`. Its document part templates are in the Word Quick Parts Gallery. For PDF, HTML, and single-file HTML output, the template libraries are in `default.pdfxtx` and `default.htmxtx` or `default.htmtdotx` template text files, respectively. Each document part template is in a separate section of the template file. The `default.pdfxtx` and `default.htmtdotx` files are zipped files packages and must be unzipped before you can edit them.

### Open a Word Template File

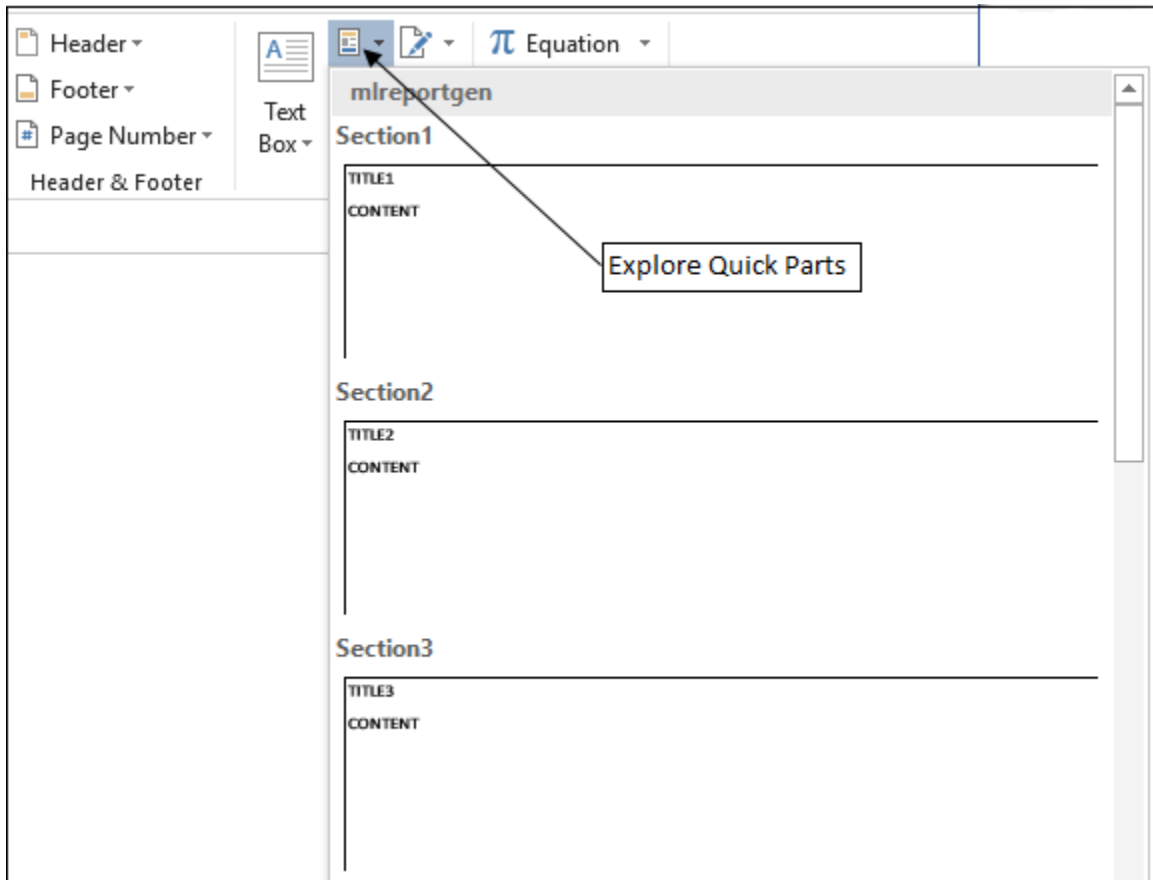
To open a Word template file, which contains a template library, right-click the `default.dotx` file name and select **Open**.

---

**Note** To open a Word template file, do not double-click its `.dotx` file name. Double-clicking its file name opens a Word document (`.docx`) file that uses the `.dotx` file as its template.

---

For some reporters, such as the Section and Chapter reporters, when you open the Word template file, a blank page appears. To see images of the individual templates, click the **Explore Quick Parts** icon in the **Text** section of the **Insert** ribbon. The document part templates for reporters are of type `mlreportgen`.



To view the contents of an individual template, click its image in the Quick Parts Gallery to copy it into the main part of the document.

To see its holes, select **Design Mode** in the Developer Ribbon. If the Developer Ribbon is not displayed, select **File Options** and then, **Customize Ribbon**. In the **Customize the Ribbon** list, check Developer.

To display paragraph and other formatting marks, select **File Options** and then, select **Display**. Check **Show all formatting marks**.

---

**Note** If you do not want to change the template file, remove all copied templates from the main part of the document before you close or save the template file.

---

## Open a PDF Template Library File

To open a PDF template library file, use `unzipTemplate` to unzip the default `.pdftx` file. Then, use any text editor to open the `docpart_template.html` template library file. The template library file has an `html` extension because PDF templates are based on HTML templates.

## **Open an HTML Template Library File**

### **Open an HTML Template Library File**

To open an HTML template library file, use `unzipTemplate` to unzip the `default.htmtx` file. Then, use any text editor to open the `docpart_template.html` template library file.

### **Open a Single-File HTML Template Library File**

To open a single-file HTML template, use any text editor to open the `default.htmt` template library file.



## Reporter Templates

This reference describes the templates used by these reporters:

- Section (see “Section Templates” on page 3-5)
- Chapter (see “Chapter Templates” on page 3-14)

Use this information to create custom templates for use with one of these reporters or from a reporter derived from these reporters.

### Section Templates

The Section reporter uses six sets of templates for each of the three output types supported by MATLAB Report Generator: docx, pdf, and html. Each set corresponds to the six levels of section hierarchy that the Section reporter can generate. The first set corresponds to a top-level section, the second set to a second-level subsection of a top-level section, and so on.

Each level set contains three templates:

- A section body template named Section $N$ , where  $N$  is the level of a section (see “Section1 Template” on page 3-5 and “Section2 - Section6 Templates” on page 3-9).
- A numbered section title template named SectionNumberedTitle $N$  (see “Numbered Section Title Templates” on page 3-12).
- An unnumbered section title template names SectionTitle $N$  (see “Unnumbered Section Title Templates” on page 3-13).

A Section reporter determines which set to use when your report program adds the reporter to a report object. For example, when your report program adds a Section reporter to a Report object, the Section reporter uses the top-level template set. A Section reporter uses the second-level template set when the report's add method adds it to the report as part of the content of a top-level section reporter. A Section reporter uses the third-level template set when the report add method adds it to the report as part of the content of a second-level reporter, etc.

#### Section1 Template

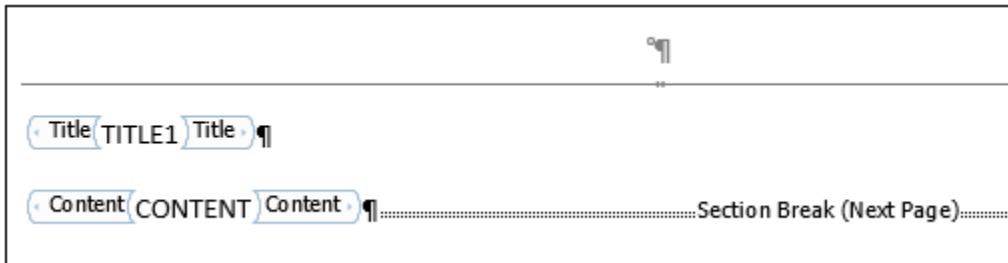
The Section reporter's Word, PDF, and HTML Section1 templates specify the format of a top-level section generated by the Section 1 reporter in a Word, PDF, and HTML report, respectively. All three templates specify the location of the holes and the default styles for the Section reporter's Title and Content properties.

In addition, the PDF and Word templates specify the page layout properties of a top-level section, including page orientation, margins, page headers, and page footers of a top-level Word or PDF section. The Word and PDF templates specify a different header for the first page of a section and for the pages that follows. The header of the first page contains only a rule. The header of subsequent pages, called default header, contains the section title. The first page footer and subsequent page footers are identical.

A top-level section starts on a new page having the properties specified by the top level Section reporter's template. All subsequent pages needed to accommodate the top-level section content have the same page layout, unless the content itself specifies a new page layout, in which case, the page layout of subsequent pages changes. Default lower-level section templates do not specify a page layout. As a result, adding subsections to a section does not change the page layout. The content of subsections has the same layout as the top-level sections.

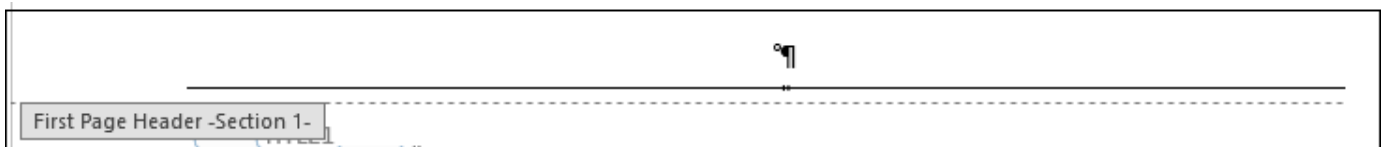
- Word Section1 Template

The Word Section1 template resides in the QuickParts gallery of the Section reporter's default.dotx template file. The QuickParts gallery serves as the Section reporter's Word template library. To view or edit the Section1 template, you must open the default.dotx file in Word and create an instance of the template in the default.dotx template. The Section1 template appears as follows in Word:



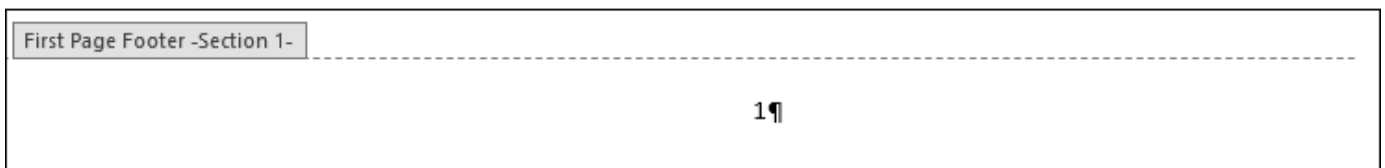
**Note** When you display a copy of the Word Section1 template, it scrolls to the default page header on the second page, which displays an error message. Refer to the note below for an explanation of this error message. To see the template holes, scroll to the top of the template.

The Section 1 template specifies a header for the first page of a section that differs from the header of succeeding pages. The header of all but the first page is called the default header. The first page header contains a rule but is otherwise empty.



**Note** The rule is implemented as an empty paragraph with a bottom border. The font size of the paragraph is set to a very low value to minimize the paragraph's height.

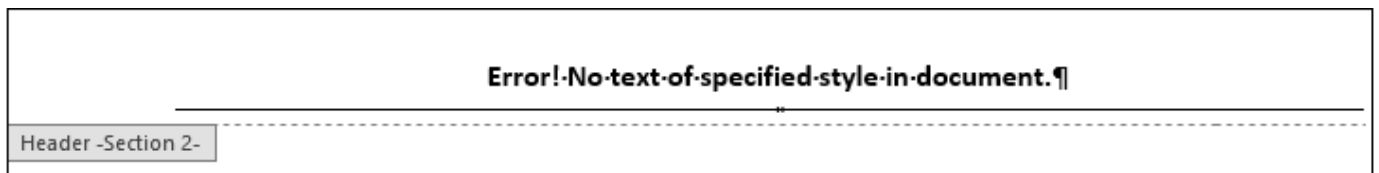
The first page footer contains a Word page number field. When this template or a report generated from it is opened in Word, Word replaces the field with the number of the page on which the footer appears.



The Section1's default page header contains a Word StyleRef field that references the section's title style (that is, SectionTitle). When the template or a report generated from the template is opened in Word, Word replaces the StyleRef field with the content of the first paragraph in the section that has the SectionTitle style. In a report, that paragraph contains the section's title.

Thus, the section's title appears in page headers that follow the first page. (This is called a running head in document design.)

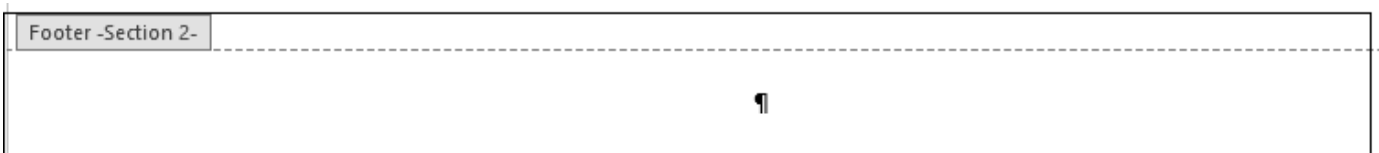
**Note** The Section1 template occupies less than a page. Thus, when you copy the Section1 template from the Section reporter's QuickPart's gallery (that is, template library) to the body of the `default.dotx` template, only the first page of the section appears. However, Word creates a page with a new section (Section 2) that inherits the Section1 template's page headers. Word replaces the StyleRef field in the header of this new section with an error message because there is no paragraph in the new section with the referenced style.



The Section1 template specifies a default page footer that is identical in content and format to the first page footer.

**Note** To view or edit the default page footer, you must insert a page in the Section1 template as follows:

- 1 Copy the template from the Section reporter template's QuickParts gallery (that is, its template library) to the body of the reporter template.
- 2 Insert a paragraph after the Content hole in the template.
- 3 Enable the paragraph's Page Break Before property.



- PDF Section1 Template

The PDF Section1 template resides in the template library of the Section reporter's PDF template file (`default.pdfctx`). The template file is a zip file. It stores the template library in a file named `docpart_templates.html`. To view or edit the Section1 template:

- 1 Unzip a copy of the `default.pdfctx` file using the Report API `unzipTemplate` command.
- 2 Open the `docpart_templates.html` file in the MATLAB editor or any other text editor.
- 3 When you are finished editing the Section1 template, save the `docpart_templates.html` file.
- 4 Rezip the `default.pdfctx` file, using the Report API `zipTemplate` command.

For more information, see "Create a PDF Document Part Template Library" on page 13-32.

The PDF Section1 template uses the following HTML markup to define the page layout of a top-level PDF section generated by the Section reporter. The markup also defines the location of the holes to be filled with the content of the Section reporter's Title and Content properties.

```
<dptemplate name="Section1">
  <layout style="page-margin: 0.5in 1in 0.5in 1in 0.5in 0.5in 0in; page-size: 8.5in 11in portrait">
    <pheader type="first" template-name="SectionFirstPageHeader"/>
    <pfooter type="first" template-name="SectionFirstPageFooter"/>
    <pheader type="default" template-name="SectionDefaultPageHeader"/>
    <pfooter type="default" template-name="SectionDefaultPageFooter"/>
    <pheader type="even" template-name="SectionEvenPageHeader"/>
    <pfooter type="even" template-name="SectionEvenPageFooter"/>
    <pnumber format="1" />
  </layout>
  <hole id="Title" default-style-name="SectionTitle1">TITLE</hole>
  <hole id="Content" default-style-name="SectionContent">CONTENT</hole>
</dptemplate>
```

The pheader and pfooter elements in the Section1 layout specify templates used to define the content and layout of a top-level section's page headers and footers. The header and footer templates reside in the same template library file (`docpart_templates.html`) as the Section1 template itself. The Section reporter uses only the first page and default page templates.

The first page header and footer templates are

```
<dptemplate name = "SectionFirstPageHeader">
  <p class="SectionTitleHeader"></p>
  <hr/>
</dptemplate>

<dptemplate name = "SectionFirstPageFooter">
  <hr/>
  <p class="SectionTitleFooter"><page/></p>
</dptemplate>
```

The header template specifies an empty paragraph followed by a horizontal rule. The empty paragraph specifies the style SectionTitleHead. It is defined in the template's style sheet (see below). The first page footer template specifies a horizontal rule followed by a page number.

The default page header template

```
<dptemplate name = "SectionDefaultPageHeader">
  <p class="SectionTitleHeader"><StyleRef style-name="SectionTitle1"/></p>
  <hr/>
</dptemplate>
```

specifies a paragraph containing a styleref followed by a horizontal rule. During report generation, the Report API replaces the styleref element with the content of the top-level section's title paragraph, thereby creating a running head.

The default page footer template

```
<dptemplate name = "SectionDefaultPageFooter">
  <hr/>
  <p class="SectionTitleFooter"><page/></p>
</dptemplate>
```

specifies a horizontal rule followed by an automatically generated page number.

The styles for the header and footer templates are in the pdf/stylesheets/root.css file.

```
p.SectionTitleHeader {
  font-family: 'Noto Sans', 'Noto Sans CJK JP', 'Noto Sans CJK
SC', 'Noto Sans CJK KR';
  font-size: 11pt;
  color: black;
  white-space: pre;
  text-align:center;
}

p.SectionTitleFooter {
  font-family: 'Noto Sans', 'Noto Sans CJK JP', 'Noto Sans CJK
SC', 'Noto Sans CJK KR';
  font-size: 11pt;
  color: black;
  white-space: pre;
  text-align:center;
}
```

- HTML Section1 Template

The Section1 document part template in the default.htmlt file specifies the Title and Content holes.

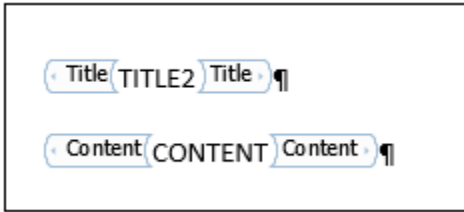
```
<dptemplate name="Section1">
  <hole id="Title" default-style-name="SectionTitle1">TITLE</hole>
  <hole id="Content">CONTENT</hole>
</dptemplate>
```

## Section2 - Section6 Templates

The Section2-Section6 templates specify the format of subsections generated by the Section reporter. Each template contains hole elements that specify the location of holes to be filled with the content of the Section reporter's Title and Content properties, respectively. The Title hole in each template specifies a default title style specific to the subsection level defined by the template.

- Word Section2 - Section6 Templates

As an example, this image shows the Section2 template.



- PDF and HTML Section2 - Section6 Templates

As an example, this image shows the Section2 template.

```
<dptemplate name="Section2">
  <hole id="Title" default-style-name="SectionTitle2">TITLE</hole>
  <hole id="Content">CONTENT</hole>
</dptemplate>
```

### Section Template Holes

All of the Section templates (Section1 - Section6) have the holes described in this table.

Hold ID	Hole Type	Description
Title	Block	<p>The Section reporter fills this hole with content based on the type of the content of its Title property as follows.</p> <ul style="list-style-type: none"> <li>• If the Title property content is an inline object, such as a string or <code>mlreportgen.domText</code> object, the Section reporter uses a <code>SectionTitle</code> reporter to format and generate the inline content as a title. It then adds the generated content to the Title hole. The <code>SectionTitle</code> reporter use templates to format the inline content.</li> <li>• If its Title property content is a paragraph or other block-level object, the Section reporter adds the paragraph to the Title hole.</li> <li>• If its Title property content is another reporter, the Section reporter adds the content generated by the reporter to the Title hole.</li> </ul>

Hold ID	Hole Type	Description
Content	Block	Content of the section

### Section Template Styles

The Section templates use styles to format some content. The Word templates define the styles they use in the style sheet in the `default.dotx` template file. The PDF and HTML templates define the styles in the `stylesheets/root.css` file in the `default.pdf` and `default.html` files, respectively. The following table describes the styles used by the Section templates.

Style Name	Style Type	Description
SectionContent	Character	The Content hole in the Section1-Section6 templates specify this style as the default text style for content that fills the hole. The content can specify styles or formats that override the default style.
SectionTitle1 - SectionTitle6	Character	The Title hole in the corresponding section template specifies the corresponding style name as the default style for the section's title. For example, the Title hole in the Section1 template specifies SectionTitle1 as the name of the default style for the title of a top-level section. Content added to the Title hole can specify formats or styles that override the default style.
SectionTitleHeader	Character	The Section1 page headers use this style to center the header content.
SectionTitleFooter	Character	The Section1 page footers use this style for the footer content.

### Section Title Templates

If the content of the Section reporter's `Title` property is a string, `Text`, or other inline object, it uses a `SectionTitle` reporter to generate the content used to fill the Title hole in its section level templates. The `SectionTitle` reporter in turn uses templates to format the inline content as a title. The `SectionTitle` reporter use two sets of templates for each output type, one to create hierarchically numbered titles (1.1, 1.2, 1.2.1, and so on), the other to create unnumbered titles. Each set contains six templates corresponding to the six levels of sections that the `Section` reporter can generate. The templates reside in the template libraries of the Section reporter's Word, PDF, and HTML template files, `default.dotx`, `default.pdf`, and `default.html`, respectively.

### Numbered Section Title Templates

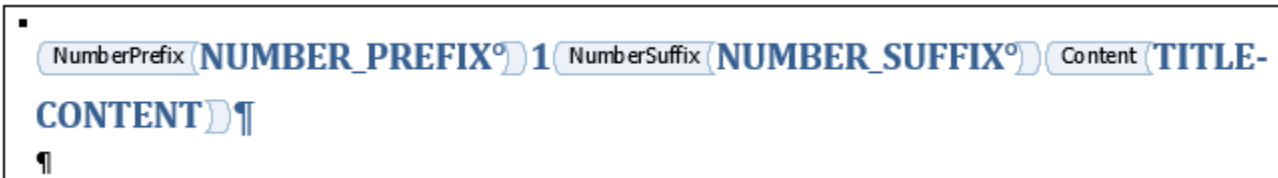
If a report or Section reporter specifies that its titles be numbered, the SectionTitle reporter uses automatically numbered templates to format the inline content of the Section reporter's Title property. The titles are named SectionNumberedTitleN, where N is the section level to which the template applies. For example, the name of the template for a top-level section title is SectionNumberedTitle1. Each template contains a paragraph element that specifies the same style as is specified by the Title hole in the corresponding section level template, for example, SectionTitle1 for a top-level section title. See "Section Template Styles" on page 3-11.

The title paragraph contains the following holes.

- NumberPrefix hole to be filled with the content of SectionTitle reporter's NumberPrefix property (empty by default)
- Autonumber markup that is replaced by a hierarchical number during report generation. The autonumber markup differs for each level template so as to generate the hierarchical number appropriate to that level.
- NumberSuffix hole to be filled with the content of the SectionTitle reporter's NumberSuffix property (empty by default).
- Content hole to be filled with the content of the SectionTitle reporter's Content property.

The following images show the Word, PDF, and HTML SectionNumberedTitle1 templates, respectively. Lower-level templates are similar.

- Word SectionNumberedTitle1 Template



- PDF SectionNumberedTitle1 Template

```
<dptemplate name="SectionNumberedTitle1">
  <h1 class="SectionTitle1" style="counter-increment:sect1;counter-reset:sect2 sect3 sect4 sect5 sect6 figure table;"
  ><hole id="NumberPrefix">NUMBER_PREFIX</hole
  ><autonumber stream-name="sect1"></autonumber
  ><hole id="NumberSuffix" >NUMBER_SUFFIX</hole
  ><hole id="Content">TITLE CONTENT</hole
  ></h1>
</dptemplate>
```

- HTML SectionNumberedTitle1 Template



```

<dptemplate name="SectionNumberedTitle1">
  <h1 class="SectionTitle1" style="counter-increment:sect1;counter-reset:figure table;"
  ><hole id="NumberPrefix">NUMBER_PREFIX</hole
  ><autonumber stream-name="h1" format="n"></autonumber
  ><hole id="NumberSuffix" >NUMBER_SUFFIX</hole
  ><hole id="Content">TITLE CONTENT</hole
  ></h1>
</dptemplate>

```

During report generation, the Section reporter sets the Content property of the SectionTitle reporter to the inline content of the Section reporter's Title property. It does not set the NumberPrefix and NumberSuffix properties. As a result, the title generated by the SectionTitle reporter consists by default of a hierarchical number followed by the title text.

The SectionTitle reporter provides the NumberPrefix and NumberSuffix holes to facilitate labeling of titles by derived reporters. For example, the Chapter reporter, which is derived from the Section reporter, sets the NumberPrefix property to Chapter in English locales. In some East Asian locales, the Chapter reporter sets the NumberSuffix to the character designating chapter.

---

**Note** If you customize a numbered section template, do not remove or replace the SEQ fields in a Word template or the autonumber markup in a PDF or HTML template. To generate unnumbered sections, use the “Unnumbered Section Title Templates” on page 3-13.

---

### SectionNumberedTitle Template Holes

All of the SectionNumberedTitle templates (Section1 - Section6) have the holes described in this table.

Hole ID	Hole Type	Description
NumberPrefix	Inline	Prefix to display before the section number.
NumberSuffix	Inline	Suffix to display after the section number.
Content	Inline	Content of the title

### Unnumbered Section Title Templates

If the report specifies that the current section uses unnumbered titles, the SectionTitle reporter uses unnumbered templates to generate section titles. The unnumbered templates are named SectionTitle $N$  where  $N$  is the level of the section whose title is to be generated. For example, the template for a top-level section is named SectionTitle1. Each template contains a paragraph element that specifies the same style as is specified by the Title hole in the corresponding section level template, for example, SectionTitle1, for a top-level section title. See “Section Template Styles” on page 3-11. The title paragraph contains a hole to be filled by the content of the SectionReporter's Content property (set by the Section reporter during report generation).

The follow images show the Word, PDF, and HTML versions of the SectionTitle1 templates. Lower-level templates are similar.

- Word SectionTitle Template

All levels of the Word SectionTitle templates have the same content hole.



- PDF and HTML SectionTitle1 Template

```
<dptemplate name="SectionTitle1">
  <h1 class="SectionTitle1"
    ><hole id="Content">title text</hole
  ></h1>
</dptemplate>
```

- PDF and HTML SectionTitle2 - SectionTitle6 Templates

These section title templates include a Content hole, as in SectionTitle1 template. Each of these sections specifies its title style.

```
<dptemplate name="SectionTitle2">
  <h2 class="SectionTitle2"
    ><hole id="Content" default-style-name="SectionTitle1">title text</hole
  ></h2>
</dptemplate>
```

### SectionTitle Template Hole

All of the SectionTitle templates (Section1 - Section6) contain the hole described in this table.

Hole ID	Hole Type	Description
Content	Inline	Content of the title

## Chapter Templates

The Chapter reporter, a subclass of the Section reporter, uses the Section reporter's top-level template set to generate its content. This is because a Chapter-generated section is nearly identical to a Section-generated section. However, the two types of sections differ in two respects:

- The title of a chapter section contains the word Chapter in English locales or the equivalent in other locales supported by the Report API. The Chapter reporter includes the word Chapter in titles by setting the NumberPrefix or NumberSuffix properties of the SectionTitle reporter used to generate the chapter title.
- All Section-generated top-level sections start on page 1. By contrast, only the first chapter generated by a Chapter reporter starts on page 1. Succeeding chapters continue page numbering from the previous chapter. The Chapter reporter implements this behavior programmatically, thus avoiding the need to use a modified version of the Section top-level template.

See "Section1 Template" on page 3-5, "Numbered Section Title Templates" on page 3-12, and "Unnumbered Section Title Templates" on page 3-13

## Add Report Explorer Content to a Report

Use the Report API's `RptFile` reporter to add Report Explorer content to a Report API report. For example, this script adds content generated by a Report Explorer setup file named `recontent.rpt` to a report generated by the Report API.

```
import mlreportgen.report.*

rpt = Report('myreport', 'docx');
append(rpt, TitlePage("Title", "My Report"));

append(rpt, RptFile( "recontent.rpt"));

close(rpt);
rptview(rpt);
```

---

**Note** To work with the Report API, a Report Explorer setup file must conform to certain restrictions. See `mlreportgen.report.RptFile` for more information.

---

## Customize Chapters

You can customize the content and format of chapters generated by the Report API Chapter reporter, `mlreportgen.report.Chapter`. You can perform some customizations by setting properties. For example, you can use the Chapter reporter `Layout` property to change the chapter layout from portrait to landscape. For more information on chapter customizations that are available by setting properties, see `mlreportgen.report.Chapter`.

For other customizations, you must modify the Chapter reporter template, class definition file, or both the template and class definition file. To customize only static content, such as a logo in a page header, modify the Chapter reporter default template and use the modified template with the reporter. To customize dynamic content or both dynamic and static content, create a custom Chapter reporter and modify its template and class definition file.

For an example that customizes chapter page headers with fixed and dynamic content, see “Customize Chapter Page Headers” on page 3-24.

### Customize Static Content

If your chapter customization includes only static content, create a copy of a chapter template, edit the copy, and use the new template when you generate a report.

#### Create the Template

Create a copy of the chapter template for the type of report that you plan to generate. For example, to copy the template for a PDF chapter to the file `mychapter.pdf` in the folder `mytemplates`, enter:

```
mlreportgen.report.Chapter.createTemplate('mytemplates/mychapter', 'pdf');
```

#### Edit the Template

The way that you edit a chapter template depends on the type of report that you are generating:

- “Edit a PDF Template” on page 3-19
- “Edit a Word Template” on page 3-20

#### Use the Template

In your report generation program:

- 1 Define a chapter reporter.
- 2 Set the `TemplateSrc` property of the reporter to the path of the custom template.

For example:

```
chapter = mlreportgen.report.Chapter;
chapter.TemplateSrc = 'mytemplates/mychapter.pdf';
```

### Customize Dynamic Content

If your chapter customization includes dynamic content, you must create a skeleton custom reporter class so that you can define properties for the dynamic content. When you create the skeleton custom chapter reporter, the chapter templates for all report types (PDF, Word, and HTML) are copied to a

resources/templates folder. Edit the chapter template to contain the content that you want to generate. Add holes for the dynamic content. When you generate a report, use the custom chapter reporter and assign values to the properties that correspond to the holes.

#### Create a Skeleton Custom Reporter Class

Use the `mlreportgen.report.Chapter.customizeReporter` method to:

- Create a skeleton reporter class.
- Copy the default templates for each type of report to a `resources/templates` folder.

For example, this code creates a class named `Chapter` and copies the templates to the `resources/templates` folder in the folder `+myCompany/@Chapter`:

```
mlreportgen.report.Chapter.customizeReporter( '+myCompany/@Chapter' )
```

#### Edit the Template

The way that you edit a chapter template depends on the type of report that you are generating:

- “Edit a PDF Template” on page 3-19
- “Edit a Word Template” on page 3-20

#### Define the Properties for the Dynamic Content

In the custom reporter class, define a property for each hole that you added to the template. For example:

```
classdef myChapter < mlreportgen.report.Chapter
    properties
        Project = ''
        Date = ''
    end
    ...
end
```

A property name must match a hole ID in the PDF template or a hole **Title** in the Word template.

#### Use the Custom Reporter Class

In your report generation program:

- Create a chapter reporter from the custom chapter reporter class.
- Assign values to the reporter properties that correspond to the holes in the template.

For example:

```
chapter = myCompany.Chapter();
chapter.Project = 'ABC Project';
chapter.Date = date;
```

## Edit a PDF Template

The PDF chapter template file is a zip file. To modify the template, unzip the template file, edit `docpart_templates.html` in a text editor, and package the extracted files back into the zip file. See “Chapter Templates” on page 3-14.

### Locate the Template File

If your template file was created by using `m1reportgen.report.Chapter.createTemplate`, the packaged template file is in the location that you specified. For example, this code creates the template file `mychapter.pdf` in the `mytemplates` folder:

```
m1reportgen.report.Chapter.createTemplate('mytemplates/mychapter', 'pdf');
```

If your template file was created by using `m1reportgen.report.Chapter.customizeReporter`, the template file has the name `default.pdf` and is in the `resources/templates/pdf` subfolder of the folder that contains the chapter reporter class definition file.

### Unzip the Template File

Unzip the package file by using the `unzipTemplate` function. For example, this code extracts files from the template file `mytemplates/mychapter.pdf` into the folder `mychapter`:

```
unzipTemplate('mytemplates/mychapter.pdf', 'mychapter');
```

The extracted files include:

- `docpart_templates.html`
- `root.html`
- Folders for images and stylesheets

### Edit the Markup

Open `docpart_templates.html` in a text editor and edit the markup.

To define a hole for dynamic content, use the `hole` element. When you add a property for the hole in the custom chapter reporter, the property name must match the `hole id` value.

Modify a header or footer by editing the header or footer template in the `dptemplate` element with the name `Section1`. The templates for headers and footers are `SectionFirstPageHeader`, `SectionFirstPageFooter`, `SectionDefaultPageHeader`, `SectionDefaultPageFooter`, `SectionEvenPageHeader`, and `SectionEvenPageFooter`.

To specify the same header or footer for all chapter pages, provide only the `SectionDefaultPageHeader` or `SectionDefaultPageFooter` in the list of header and footer templates. Remove the other headers and footers from the list.

---

**Note** Use the existing `dptemplate` elements. Do not create your own `dptemplate` elements or change the values of the `Name` attribute of the `dptemplate` elements.

---

### Package the Template Files

Package the files back into the template file by using the `zipTemplate` function. For example, if `mytemplates/mychapter` contains the unzipped files, this code packages the files into `mychapter.pdfctx` in the `mytemplates` folder.

```
zipTemplate('mytemplates/mychapter.pdfctx', 'mytemplates/mychapter');
```

### Edit a Word Template

To edit a Word template:

- 1 Open the template file in Word.
- 2 Create a temporary copy of the Section1 template in the template document. The Section1 template, which is used for both chapters and top-level sections, is in the **Quick Parts** gallery of the template document.
- 3 Edit the temporary copy and save the copy to the **Quick Parts** gallery.
- 4 Delete the temporary copy from the template document and save the template file.

### Locate the Template File

If your template file was created by using `mlreportgen.report.Chapter.createTemplate`, the template file is in the location that you specified. For example, this code creates the template file `mychapter.dotx` in the `mytemplates` folder:

```
mlreportgen.report.Chapter.createTemplate('mytemplates/mychapter', 'docx');
```

If your template file was created by using `mlreportgen.report.Chapter.customizeReporter`, the template file has the name `default.dotx` and is in the `resources/templates/docx` subfolder of the folder that contains the chapter reporter class definition file.

### Open the Template File in Word

Open the template file by using one of these methods:

- In MATLAB, in the Current Folder pane, right-click the template file and click **Open Outside MATLAB**.
- Outside of MATLAB, right-click the template file and click **Open**.


---

**Note** Do not double-click a Word template file to open it. Double-clicking the file opens a Word document file that uses the template.

---

The template document opens to an empty page.

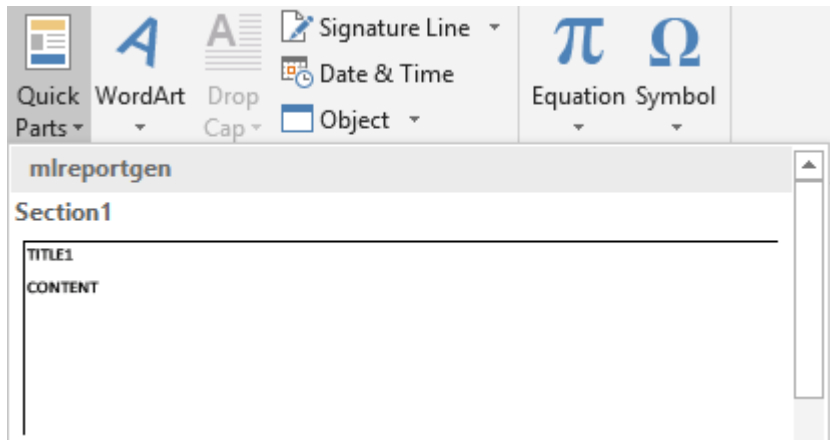
### Show the Formatting Symbols

To make the paragraph and formatting symbols visible, on the **Home** tab, click the **Show/Hide**  button.



### Copy the Section1 Template into the Template Document

- 1 Position the cursor in front of the paragraph in the template document.
- 2 On the **Insert** tab, in the **Text** group, click **Quick Parts**, then click the Section1 building block.



Word inserts a copy of the Section1 template and a dummy Section2 section. The dummy section is ignored when you generate a report. The cursor is positioned in the body of the dummy section.

- 3 Scroll up to the Section1 first page template.



- 4 Edit the template to contain the content that you want to generate. To add holes to a Word template, see “Add Holes in a Microsoft Word Template” on page 13-120.

To modify headers and footers, see “Tips for Editing Headers and Footers in a Word Template” on page 3-22.

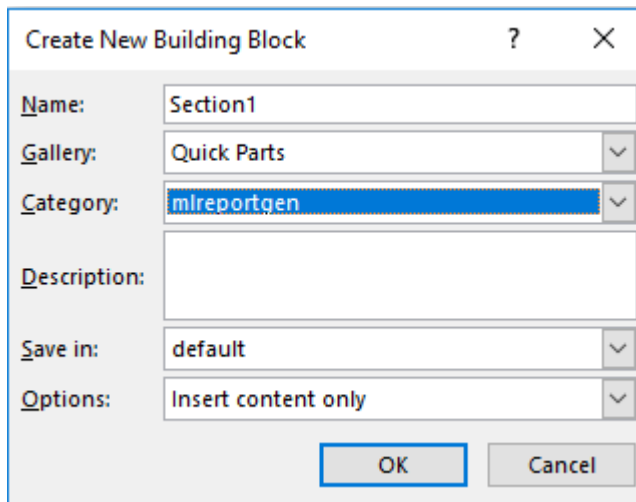
---

**Note** Use the existing templates in the Quick Parts gallery. Do not create your own templates or change the names of the existing templates.

---

### Save the Template in the Quick Parts Gallery

- 1 If the header or footer is open, close it by double-clicking the page outside of the header.
- 2 Select all of the content in the template by pressing **Ctrl+A**.
- 3 On the **Insert** tab, click **Quick Parts**, and then click **Save Selection to Quick Parts Gallery**.
- 4 In the **Create New Building Block** dialog box, in the **Name** field, enter Section1. Set **Gallery** to Quick Parts, **Category** to mlreportgen, and **Save in** to the name of the template file.



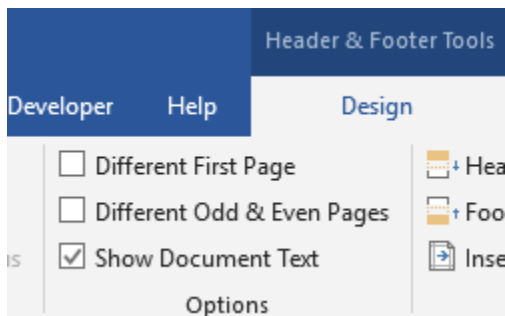
### Save the Template File

It is a best practice to delete the content from the body of the template document before you save the template file.

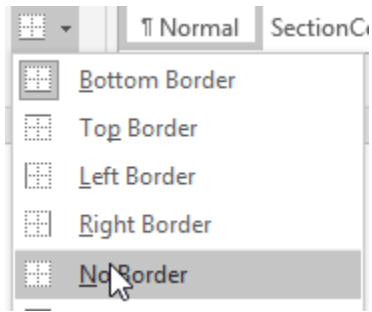
## Tips for Editing Headers and Footers in a Word Template

Here are some tips for editing headers and footers in a Word Template.

- To control whether headers and footers on the first page differ from headers and footers on the following pages, double-click a header. Then, under **Header and Footer Tools**, on the **Design** tab, select or clear the **Different First Page** check box.



- To control whether even page headers and footers differ from odd page headers and footers, select or clear the **Different Odd and even Pages** check box.
- To see the headers and footers for pages that follow page one, add empty paragraphs to page one until a second Section1 page is created. Double-click the header on the new page.
- The default first page header includes a horizontal rule, which is the bottom edge of a small paragraph. To make the rule invisible, select the small paragraph. Then, on the **Home** tab, in the **Paragraph** group, **Borders > No Border**.



## See Also

`mlreportgen.report.Chapter` | `unzipTemplate` | `zipTemplate`

## More About

- “Reporter Templates” on page 3-5
- “Open a Template File” on page 3-2
- “Add Holes in a Microsoft Word Template” on page 13-120
- “Add Holes in HTML and PDF Templates” on page 13-132
- “Customize Chapter Page Headers” on page 3-24

## Customize Chapter Page Headers

This example shows how to customize chapter page headers that are generated by the Report API chapter reporter. You can customize chapter page headers for PDF and Microsoft® Word reports. The example generates a report for a fictitious company, ABC Services. The custom header contains the company logo, project name, and report date.



Control Systems Consulting

18-Jul-2019

---

The company logo is fixed in the header. The project name and report date are dynamic. They are created when the report is generated.

The workflow is:

- 1 Create a custom chapter reporter class.
- 2 Modify the headers in the PDF or Word chapter template.
- 3 Add properties to the custom reporter class for the dynamic content in the custom headers.
- 4 Write a report program that uses the custom reporter and specifies values for the properties.

### Create a Custom Chapter Reporter Class

Create a skeleton chapter reporter class by calling the `mlreportgen.report.Chapter.customizeReporter` method. Name the class `Chapter` and save it in a class folder inside of a package folder.

```
mlreportgen.report.Chapter.customizeReporter('+abc/@Chapter');
```

This method call also copies the chapter template for each report type to the `+abc/@Chapter/resources/templates` folder.

### Modify the Headers in the Chapter Template for a PDF Report

To modify the header definitions in the chapter template for a PDF report:

- 1 Unzip the template file.
- 2 Edit the header definitions in the `docpart_templates.html` file using a text editor.
- 3 Package the extracted files into the template file.

### Unzip the Template File

Change the current folder to the folder that contains the PDF template file, `default.pdf` and then unzip the file.

```
cd('+abc/@Chapter/resources/templates/pdf');  
unzipTemplate('default.pdf');
```

The extracted files are in a folder named `default`.

## Copy the Image File

Copy the image file for the company logo to the default/images folder. Use the image file, abc\_logo.png, attached to this example, or your own image file.

## Edit the Header Definitions

In the MATLAB® editor or a text editor, open docpart\_templates.html, which is in the folder named default.

To make the headers the same on all pages, in the Section1 dptemplate element, delete SectionFirstPageHeader and SectionEvenPageHeader from the list of header and footer templates.

```
<dptemplate name="Section1">
  <layout style="page-margin: 0.5in lin 0.5in lin 0.5in 0.5in 0in; page-size: 8.5in llin portrait">
    <pheader type="first" template-name="SectionFirstPageHeader"/>
    <pfooter type="first" template-name="SectionFirstPageFooter"/>
    <pheader type="default" template-name="SectionDefaultPageHeader"/>
    <pfooter type="default" template-name="SectionDefaultPageFooter"/>
    <pheader type="even" template-name="SectionEvenPageHeader"/>
    <pfooter type="even" template-name="SectionEvenPageFooter"/>
    <pnumber format="1" />
  </layout>
  <hole id="Title" default-style-name="SectionTitle1">TITLE</hole>
  <hole id="Content" default-style-name="SectionContent">CONTENT</hole>
</dptemplate>
```

SectionDefaultPageHeader is now the template for all chapter page headers. Find the SectionDefaultPageHeader template.

```
<dptemplate name = "SectionDefaultPageHeader">
  <p class="SectionTitleHeader"><StyleRef style-name="SectionTitle1"/></p>
  <hr/>
</dptemplate>
```

Replace it with this markup:

```
<dptemplate name = "SectionDefaultPageHeader">
  <table width="100%">
    <tr style="border-bottom: 1pt solid #ccc;">
      <td style="text-align:left; padding-bottom:2pt" valign="bottom">
        </img></td>
      <td style="text-align:center; padding-bottom:2pt; font-family:arial;
        font-size: 10pt; font-weight: bold" valign="bottom">
        <hole id="Project">PROJECT</hole></td>
      <td style="text-align:right; padding-bottom:2pt; font-family:arial;
        font-size: 10pt" valign="bottom"><hole id="Date">DATE</hole></td>
    </tr>
  </table>
</dptemplate>
```

The markup defines a one-row, three-column table with these characteristics:

- The table cells contain the company logo, a hole for the project name, and a hole for the report date.

- The contents of the first, second, and third cells are left-aligned, center-aligned, and right-aligned, respectively.
- The padding between the bottom of the cell and the cell contents is 2pt.
- The cell contents are aligned with the bottom of the cell.
- The font for the text in the second cell is Arial, 10pt, and bold. The font for the text in the third cell is Arial and 10pt.
- The bottom border of the row is visible. The table does not have borders because the HTML does not specify the `border` property.

#### Package the Extracted Files

If you edited `docpart_templates.html` in the MATLAB Editor, close the file before you package the files.

Make sure that you are in the `+abc/@Chapter/resources/templates/pdf` folder, which contains the folder named `default`. Package the files in `default` back into the original PDF template file.

```
zipTemplate('default.pdf', 'default');
```

#### Modify the Headers in the Chapter Template for a Word Report

In Word, the `Section1` template is used for chapters and top-level sections. The template is in the **Quick Parts** gallery.

To modify the headers in the `Section1` template:

- 1 Open the template file in Word.
- 2 Create a temporary copy of the `Section1` template in the body of the template document.
- 3 Specify whether all pages have the same header.
- 4 Edit the headers.
- 5 Save the modified `Section1` template to the **Quick Parts** gallery.
- 6 Delete the content from the body of the template document and save the template file.

#### Open the Template File

Navigate to `+abc/@Chapter/resources/templates/docx`.

Open the template file by using one of these methods:

- In MATLAB, in the **Current Folder** pane, right-click the template file and click **Open Outside MATLAB**.
- Outside of MATLAB, right-click the template file and click **Open**.

Do not double-click a Word template file to open it. Double-clicking the file opens a Word document file that uses the template.

The template document opens to an empty page.

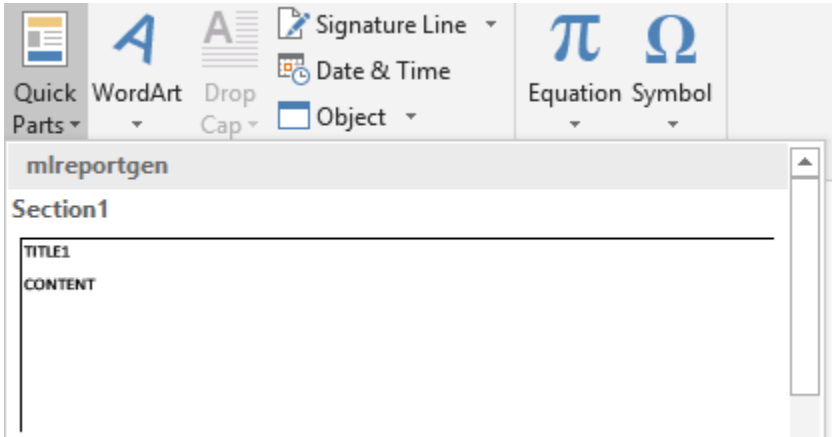
#### Display Formatting Symbols

To make paragraph and formatting symbols visible, on the **Home** tab, click the **Show/Hide** button



### Copy the Section1 Template into the Template Document

On the **Insert** tab, in the **Text** group, click **Quick Parts**, and then click the Section1 building block.



In the template document, Word inserts a copy of the Section1 template and a dummy Section2 section. The dummy section is ignored when you generate a report.

**Error!-No-text-of-specified-style-in-document.¶**

¶

The cursor is in the dummy Section2. Scroll up to the Section1 template. To open the header, double-click it.

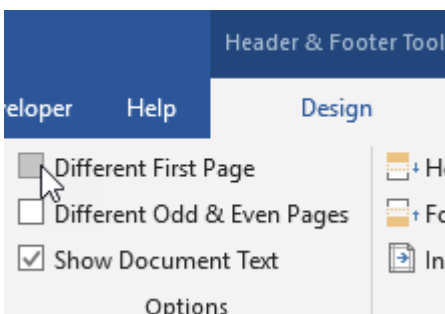
¶

First Page Header -Section 1-

CONTENT¶.....Section Break (Next Page).....

### Specify That All Pages Have the Same Header

Under **Header and Footer Tools**, on the **Design** tab, clear the **Different First Page** check box.



The header for pages that follow the first page is copied to the first page header.

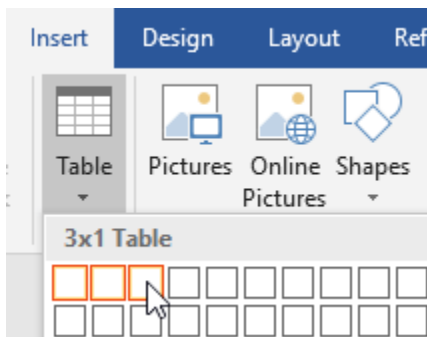
**Error! No text of specified style in document.**

Header -Section 1- E1

CONTENT ..... Section Break (Next Page) .....

### Add a Table to the Header

On the **Insert** tab, Click **Table**. Move the pointer over the grid until you highlight three columns and one row.



Delete the message Error! No text of specified style in document and the paragraph that contains it.

The horizontal rule from the original header is the bottom border of a paragraph. To make it invisible, click the paragraph and then, on the **Home** tab, in the **Paragraph** group, click the arrow to the right of **Borders** and then click **No Border**.

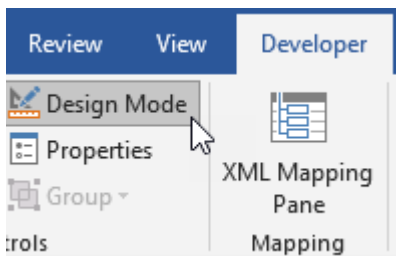
### Add the Logo to the First Table Cell

Copy and paste the company logo into the first table cell. Use the attached abc\_logo.png file or your own image file.

### Add Holes for the Project and Date

If the **Developer** ribbon is not available, click **File > Options**, and then click **Customize Ribbon**. Under **Customize the Ribbon**, select **Developer**.

On the **Developer** tab, click **Design Mode** so that you can see hole marks with the title tag when you create a hole.





In the second cell, add a few spaces before the entry mark so that the hole that you add is an inline hole.

Click in front of the spaces.

Click the **Rich Text Content Control** button . A rich text content control displays.



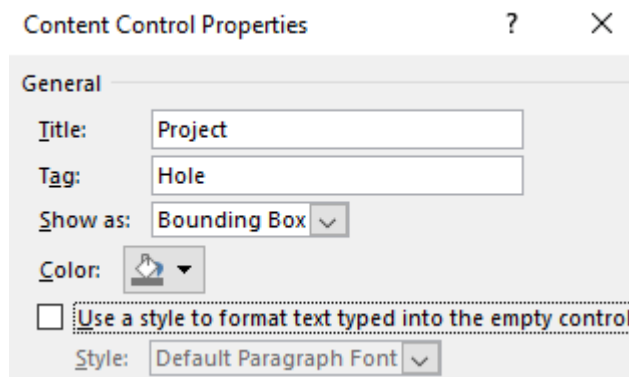
Delete the spaces that you added.

Replace the text in the control with text that identifies the hole, for example, **Project Name**.

On the **Developer** tab, click **Properties**.

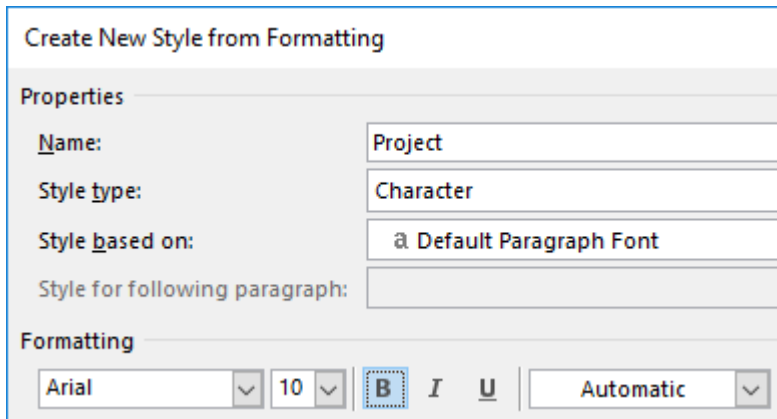
In the **Content Control Properties** dialog box:

- in the **Title** field, enter **Project**.
- In the **Tag** field, enter **Hole**.



Select the **Use a style to format text typed into the empty control** check box. Then, click **New Style**.

In the **Create New Style from Formatting** dialog box, enter **Project** in the **Name** field. For the formatting, specify **Arial, 10pt**. Click the **Bold** button. Click **OK**.



In the **Content Control Properties** dialog box, click **OK**.

Add an inline hole for the date in the third cell. Use the steps that you used to add a hole to the second cell.

- 1 Add a few spaces before the entry mark.
- 2 Click in front of the spaces.
- 3 Click the **Rich Text Content Control** button **Aa**.
- 4 Delete the spaces that you added.
- 5 Replace the text in the control with text that identifies the hole, for example, Report Date.
- 6 On the **Developer** tab, click **Properties**. In the **Content Control Properties** dialog box, enter Date in the **Title** field and Hole in the **Tag** field.
- 7 Select the **Use a style to format text typed into the empty control** check box. Then, click **New Style**.
- 8 In the **Create New Style from Formatting** dialog box, enter ReportDate in the **Name** field. For the formatting, specify Arial and 10pt.

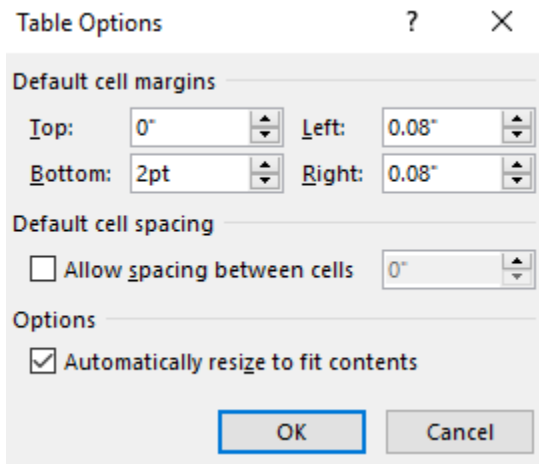
The table looks like:

		
---	---	---

### Specify the Table Cell Bottom Margin

To specify the cell bottom margin:

- 1 Select and right-click the table.
- 2 Click **Table Properties**, then on the **Table** tab, click **Options**.
- 3 In the **Table Options** dialog box, under **Default cell margins**, in the **Bottom** field, enter 2pt.

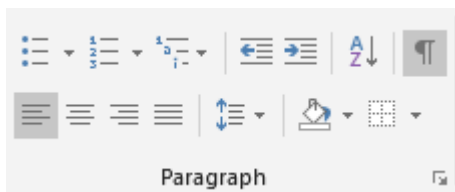


### Align the Cell Contents

Left-align the logo in the first cell. Center the hole in the third cell. Right-align the hole in the third cell.

To align cell contents:

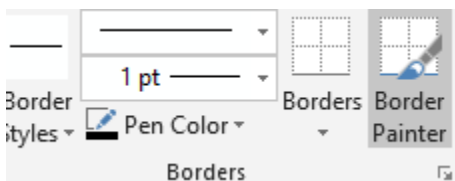
- 1 Select the image or hole in the table.
- 2 On the **Home** tab, in the **Paragraph** group, click the **Align Left**, **Center**, or **Align Right** button.



### Make the Top and Side Borders of the Table Invisible

To display only the bottom border of the table:

- 1 Select the table.
- 2 Under **Table Tools**, on the **Design** tab, click **Borders > No Border**.
- 3 Click **Borders > Bottom Border**.
- 4 Set the **Line Weight** to 1pt.



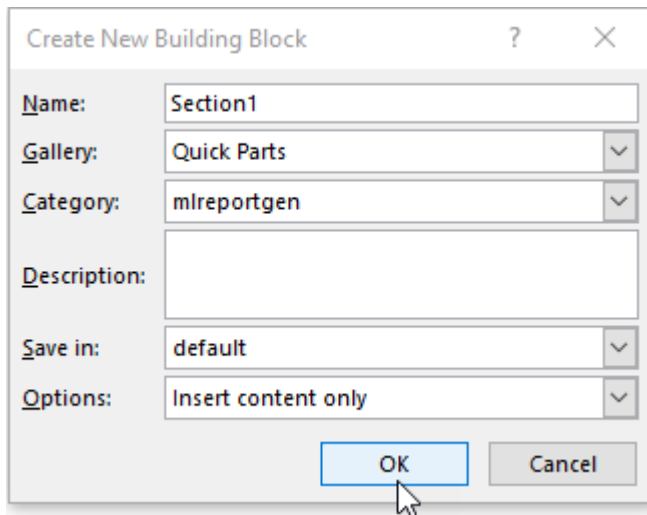
### Save the Custom Template

Close the header by double-clicking the page outside of the header.

To select all of the content in the template, press **Ctrl+A**.

On the **Insert** tab, click **Quick Parts**, and then click **Save Selection to Quick Parts Gallery**.

In the **Create New Building Block** dialog box, in the **Name** field, enter Section1. Set **Gallery** to Quick Parts, **Category** to mlreportgen, and **Save in** to default. Click **OK**.




When you see the message Do you want to redefine the building block entry?, click **Yes**.

### Save the Template File

It is a best practice to delete the content from the body of the template document before you save the template file.

With the content selected, press **Delete**.

To hide the formatting symbols, on the **Home** tab, in the **Paragraph** group, click the **Show/Hide** button .

Save the template file.

### Add Properties to the Custom Reporter Class

Because the custom header has holes for dynamic content, you must define properties for the holes.

Navigate to the +abc/@Chapter folder.

In the class definition file, Chapter.m, add the properties that correspond to the Project and Date holes in the header. Replace the empty properties section with:

```
properties
    Project = ''
    Date = ''
end
```

Save the class file.

## Generate the Report Using the Custom Chapter Reporter

When you generate the report, create the chapter reporter from the `abc.Chapter` class. Assign values to the properties that correspond to the holes in the header.

Before running the following code, navigate to the folder that contains the `+abc` folder. Alternatively, to add the folder that contains the `+abc` folder to the MATLAB path:

- 1 In the MATLAB Toolstrip, on the **Home** tab, in the **Environment** group, click **Set Path**.
- 2 In the **Set Path** dialog box, click **Add Folder**.
- 3 In the **Add Folder to Path** dialog box, click the folder and then click **Select Folder**.

You cannot add packages to the MATLAB path. Add the folder that contains the package folder.

### Generate a PDF Report

```
import mlreportgen.dom.*
import mlreportgen.report.*

report = Report("Consulting Report","pdf");
chapter = abc.Chapter();
chapter.Project = "Control Systems Consulting";
chapter.Date = date;
chapter.Title="Overview";
add(chapter,"Chapter content goes here.");
add(report,chapter);
close(report);
rptview(report);
```

### Generate a Word Report

```
import mlreportgen.dom.*
import mlreportgen.report.*

report = Report("Consulting Report","docx");
chapter = abc.Chapter();
chapter.Project = "Control Systems Consulting";
chapter.Date = date;
chapter.Title="Overview";
add(chapter,"Chapter content goes here.");
add(report,chapter);
close(report);
rptview(report);
```

## See Also

`mlreportgen.report.Chapter` | `unzipTemplate` | `zipTemplate`

## More About

- “Customize Chapters” on page 3-17
- “Reporter Templates” on page 3-5
- “Add Holes in a Microsoft Word Template” on page 13-120
- “Add Holes in HTML and PDF Templates” on page 13-132

## Create Report API Base Tables

To generate tables that have titles, create the tables by using `mlreportgen.report.BaseTable` objects. You can also divide a table created as a `BaseTable` object into legible slices by using the properties of the object. You can use the `BaseTable` object only in Report API-based reports.

If the value of the `Title` property of the `BaseTable` object consists of inline content, the table titles are numbered. The numbering scheme depends on whether the `BaseTable` object is added to a report or a chapter. The numbering scheme in a chapter depends on whether the chapter is numbered or unnumbered. For more information, see the `Title` property of the `mlreportgen.dom.BaseTable` class.

For information about other types of tables that you can create in reports, see “Choose Type of Table to Create” on page 13-62.

### Generate Tables with Numbered Titles

This example shows how to generate tables that have numbered titles. The example generates a report that has two tables with titles.

**Table 1. Magic Square with Order 3**

8	1	6
3	5	7
4	9	2

**Table 2. Magic Square with Order 4**

16	2	3	13
5	11	10	8
9	7	6	12
4	14	15	1

Import the Report API and the DOM API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
import mlreportgen.report.*
```

Create a Report API report.

```
rpt = Report('myTitledReport', 'pdf');
```

Create `mlreportgen.report.BaseTable` reporters from magic squares and specify the titles.

```
bt1 = BaseTable(magic(3));
bt1.Title = 'Magic Square with Order 3';
bt2 = BaseTable(magic(4));
bt2.Title = 'Magic Square with Order 4';
```

Add the `BaseTable` objects to the report. Close and view the report.

```
add(rpt, bt1);
add(rpt, bt2);
close(rpt);
rptview(rpt);
```

## Format BaseTable Tables

By default, a `BaseTable` object generates a table with a grid style and a bold title. To customize the table, you can use the same approaches that you use with other types of tables. You can:

- Format the DOM objects or DOM table objects before you use them to create the `BaseTable` object.
- Create the title as a DOM object. Format the DOM object before you assign it to the `Title` property of the `BaseTable` reporter.
- Set the `TableStyleName` of the `BaseTable` reporter to a custom style. The style must be defined in the template of the report to which the reporter is added or in the template of a reporter added to the report.
- Format the table content after table creation. Access the content from the `Content` property of the `BaseTable` table.

See “Format Tables” on page 13-72.

### Format an `mreportgen.dom.BaseTable` Table and Title

This example shows how to format a DOM table before you create an `mreportgen.dom.BaseTable` reporter from the table. The example also shows how to format the title of a `BaseTable` table. The example generates a table with a light blue background and a title that is not bold.

**Table 1. Magic Square with Order 3**

8	1	6
3	5	7
4	9	2

Set up the report.

```
import mreportgen.dom.*
import mreportgen.report.*
rpt = Report('myCustomBaseTable', 'pdf');
```

Create a DOM table. For example, create an informal table by using an `mreportgen.dom.Table` object.

```
t = Table(magic(3));
```

Format the DOM table. For example, make the background color of the table light blue.

```
t.TableEntriesStyle = {BackgroundColor('lightsteelblue')};
```

Create a `BaseTable` reporter from the `Table` object.

```
bt = BaseTable(t);
```

Create an `mlreportgen.dom.Text` object for the title and format it. This example makes the weight of the title text regular instead of bold.

```
tabletitle = Text('Magic Square with Order 3');  
tabletitle.Bold = false;
```

Set the `Title` property of the `BaseTable` object.

```
bt.Title = tabletitle;
```

Add the `BaseTable` object to the report.

```
add(rpt, bt);  
  
close(rpt);  
rptview(rpt);
```

## Fit Wide BaseTable Tables in a Report

If a `BaseTable` table is too wide to be legible when scaled to fit a page, you can divide the table into legible slices by setting the `MaxCols` property to the maximum number of columns to display per slice.

You can control the style of the title of the table slice by using the `TableSliceTitleStyleName` property.

## See Also

`mlreportgen.report.BaseTable`

## More About

- “Choose Type of Table to Create” on page 13-62
- “Format Tables” on page 13-72



## Update a Table of Contents or Generated List in a Word Document

When you generate a Microsoft Word report that includes a table of contents or a list of figures, tables, or other report elements, the report generation software generates a placeholder for the table of contents or list. To generate the items in the table of contents or list, you must update the Word document programmatically or interactively. If you open the Word document without updating it, the table of contents and generated lists appear to be empty.

### Update a Word Document Programmatically

To update a Word document programmatically, you can use `rptview` or `docview`.

To update and display the document, use `rptview`. If you later open the document in Word, the table of contents and other generated lists are visible. You can use `rptview` on a Windows® or Macintosh platform. Here is an example of a report generation program that uses `rptview`:

```
import mlreportgen.report.*
rpt = Report('myreport', 'docx');
add(rpt, TitlePage('Title', 'My Report'));
add(rpt, TableOfContents);
add(rpt, Chapter('Title', 'Introduction', 'Content', 'Hello World'));
close(rpt);
rptview(rpt);
```

On a Windows platform, to update the document without displaying it, use `docview`. Here is an example of a report generation program that uses `docview`:

```
import mlreportgen.report.*
rpt = Report('myreport', 'docx');
add(rpt, TitlePage('Title', 'My Report'));
add(rpt, TableOfContents);
add(rpt, Chapter('Title', 'Introduction', 'Content', 'Hello World'));
close(rpt);
docview(rpt.OutputPath, 'updatefields', 'closedoc');
```

### Update a Word Document Interactively

To update a Word document interactively, select all of the document contents and then update the selected contents. You can use these keyboard and function key shortcuts:

- On a Windows platform, press **Ctrl+A** and then press **F9**.
- On a Macintosh platform, press **Command+A** and then press **F9**.

### See Also

`rptview` | `rptview` | `docview` | `mlreportgen.report.TableOfContents` | `mlreportgen.dom.TOC` | `mlreportgen.report.ListOfFigures` | `mlreportgen.dom.LOF` | `mlreportgen.report.ListOfTables` | `mlreportgen.dom.LOT` | `mlreportgen.report.ListOfCaptions` | `mlreportgen.dom.LOC`

### **More About**

- “Table of Contents or Generated List Is Empty in a Word Report” on page 3-39
- “Create a Table of Contents” on page 13-92

# Table of Contents or Generated List Is Empty in a Word Report

## Issue

If your report generation program adds a tables of contents or a list of tables, figures, or captions to a Microsoft Word document, the report generation software creates a placeholder for the table of contents or list. If the report generation program does not include a function call to update the Word document, the entries in the table of contents or list are not generated. When you open the generated document in Word, the table of contents, list of figures, list of tables, or list of captions appears to be empty.

## Possible Solutions

Update the Word document using one of these approaches:

- In the report generation program, add a function call to update the Word document.
- Open the document in Word and then update it interactively.

For detailed information and examples, see “Update a Table of Contents or Generated List in a Word Document” on page 3-37.

## See Also

`mlreportgen.report.TableOfContents` | `mlreportgen.dom.TOC` |  
`mlreportgen.report.ListOfFigures` | `mlreportgen.dom.LOF` |  
`mlreportgen.report.ListOfTables` | `mlreportgen.dom.LOT` |  
`mlreportgen.report.ListOfCaptions` | `mlreportgen.dom.LOC`

## More About

- “Create a Table of Contents” on page 13-92
- “Update a Table of Contents or Generated List in a Word Document” on page 3-37

## Create a List of Figures or Tables in a Report API Report

In a Report API report, you can create a list of the figures or tables in a report so that you can browse the list and easily navigate to a particular figure or table. To create a list of figures, add an `mlreportgen.report.ListOfFigures` reporter to the report. To create a list of tables, add an `mlreportgen.report.ListOfTables` reporter to the report.

---

**Note** To create a section for a list of other report elements, such as equations, use an `mlreportgen.report.ListOfCaptions` reporter.

---

A `ListOfFigures` or `ListOfTables` reporter creates a placeholder for a list in a new section of the report. Typically, you add a list of figures or a list of tables section after the table of contents and before the sections that contain the report content. In PDF and Microsoft Word reports, the list of figures or tables sections are located where you put them in the report. In HTML reports, the sections are located in a sidebar.

The way the list of figures or tables is generated depends on the report type:

- PDF — The Report API generates the list during report generation.
- Word — You must generate the list by updating the Word document in your report generation program or interactively in Word. See “Update a Table of Contents or Generated List in a Word Document” on page 3-37.
- HTML — The HTML browser generates the list.

A list of figures section includes a list of the captions of all of the images that are represented as `mlreportgen.report.FormalImage` or `mlreportgen.report.Figure` objects. A list of tables section includes a list of the titles of all of the tables that are represented as `mlreportgen.report.BaseTable` objects. Each list item in a list of figures or tables links to a caption or title in the report. In PDF and Word reports, a list item also includes the page number of the caption or title in the report. A leader fills the space between the caption or title and the page number.

### Create a List of Figures Section

To create a list of figures section in a report:

- 1 Create a Report API report.
- 2 Create a `ListofFigures` object. The default title for the list of figure section is `List of Figures`. To specify a different title, set the `Title` property of the `ListofFigures` object. Add the `ListofFigures` object to the report.
- 3 Create chapters or sections for the report content.
- 4 Add report content to the chapters or sections. Create images as `FormalImage` or `Figure` objects. Specify the captions for images or figures. For a `FormalImage` object, set the `Caption` property. For a `Figure` object, set the `Caption` property of the `FormalImage` object that is specified by the `Snapshot` property of the `Figure` object.
- 5 Close the report.

## Create a List of Tables Section

To create a list of tables section in a report:

- 1 Create a Report API report.
- 2 Create a `ListofTables` object. The default title for the list of tables section is `List of Tables`. To specify a different title, set the `Title` property of the `ListofTables` object. Add the `ListofTables` object to the report.
- 3 Create chapters or sections for the report content.
- 4 Add report content to the chapters or sections. Create tables as `BaseTable` objects and specify the titles using the `Title` property of the objects.
- 5 Close the report.

## Create a Report That Has a List of Figures and a List of Tables

This example shows how to add a list of figures section and a list of tables section to a Report API report. The example generates a report about magic squares that includes two figures and a table.

Import the Report and DOM API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
import mlreportgen.dom.*
```

Create a Report API report. Add a title page and table of contents to the report.

```
rpt = Report('magic', 'docx');
tp = TitlePage;
tp.Title = 'Magic Squares';
tp.Author = 'John Doe';
append(rpt, tp);
append(rpt, TableOfContents);
```

Add a list of figures section and a list of tables section to the report. Use the default section titles.

```
lof = ListOfFigures();
append(rpt, lof);
lot = ListOfTables();
append(rpt, lot);
```

Create the report content. For this example, create a chapter for an introduction to magic squares and a chapter for an example of a magic square.

```
ch1 = Chapter('Introduction');
sec1 = Section('What is a Magic Square?');
para = Paragraph(['A magic square is an N-by-N matrix '...
'constructed from the integers 1 through N^2 '...
'with equal row, column, and diagonal sums.']);
append(sec1, para);
append(ch1, sec1);

sec2 = Section('Albrecht Durer and the Magic Square');
para = Paragraph(['A 4-by-4 magic square appears in the upper right corner '...
'of the engraving, Melancholia I, by German artist Albrecht Durer (1471-1528).']);
```

```
append(sec2,para);
append(ch1,sec2) ;
```

Add an image `durer.png` to the chapter as an `mlreportgen.report.FormalImage` object so that the image is included in the list of figures.

```
im = FormalImage('durer.png');
im.Caption = 'Melancholia I by Albrecht Durer';
append(sec2,im);
append(rpt,ch1);
```

Create the second chapter.

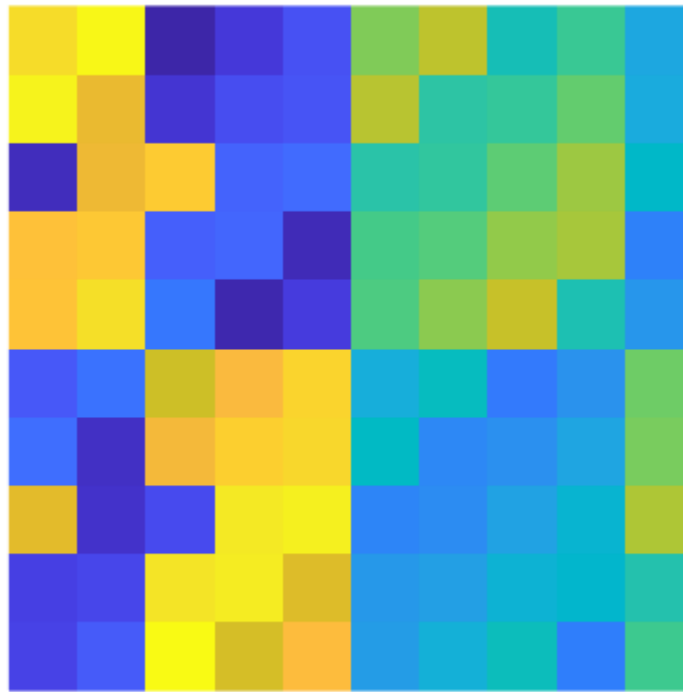
```
ch2 = Chapter('Magic Square Example');
square = magic(10);
```

Create a table from the output of `magic(10)` and add it to the chapter. Represent the table as an `mlreportgen.report.BaseTable` so that the table is included in the list of tables.

```
tbl = BaseTable(square);
tbl.Title = 'Order Ten Magic Square';
append(ch2,tbl);
```

Create a color-coded image of the magic square and add it to the chapter as an `mlreportgen.report.Figure` object so that the image is included in the list of figures.

```
clf;
imagesc(square)
axis equal
axis off
fig = Figure(gcf);
fig.Snapshot.Caption = 'Color-coded image of 10-by-10 magic square';
append(ch2,fig);
append(rpt,ch2);
```



Close and view the report.

```
close(rpt);
rptview(rpt);
```

Here is the list of figures in the report:

## List of Figures

Figure 1.1. Melancholia I by Albrecht Durer .....	1
Figure 2.1. Color-coded image of 10-by-10 magic square .....	2

Here is the list of tables in the report:

## List of Tables

Table 2.1. Order Ten Magic Square.....	2
--	---

## Customize a List of Figures or Tables Section Title

You can specify the title of a list of figures or list of tables section by setting the `Title` property of the `ListOfFigures` or `ListOfTables` reporter. To customize the style of the title, such as the font family, color, or size, use one of these approaches:

- Specify the title as an `mreportgen.dom.Text` object and specify the style using the object properties.
- Specify the title style in the template used by the `ListOfFigures` or `ListOfTables` reporter.
- Specify the title style in the template used by the title reporter for the list of figures or tables section title. See “Use a Custom Template for a List of Figures Title” on page 11-245 and “Use a Custom Template for a List of Tables Title” on page 11-252.

### See Also

`mreportgen.report.BaseTable` | `mreportgen.report.ListOfFigures` |  
`mreportgen.report.ListOfTables` | `mreportgen.report.ListOfCaptions` |  
`mreportgen.report.FormalImage` | `mreportgen.report.Figure`

### More About

- “Create a List of the Captions or Titles of Related Report Elements in a Report API Report” on page 3-46
- “Update a Table of Contents or Generated List in a Word Document” on page 3-37
- “Create a List of Figures, Tables, or Other Report Elements in a DOM API Report” on page 3-45



## Create a List of Figures, Tables, or Other Report Elements in a DOM API Report

To create a list of figures in a DOM report, use an `mlreportgen.dom.LOF` object. To create a list of tables, use an `mlreportgen.dom.LOT` object. For a list of other types of report elements, such as equations, use an `mlreportgen.dom.LOC` object.

To use a DOM LOF, LOT, or LOC object, you must create a paragraph for the caption or title of the image, table, or other report element. Then, associate the paragraph with an automatic numbering stream name that matches the automatic numbering stream name of the LOF, LOT, or LOC object. For more information and examples, see the `mlreportgen.dom.LOF`, `mlreportgen.dom.LOT`, and `mlreportgen.dom.LOC` reference pages.

Although you can also use a DOM LOF, LOT, or LOC object in a Report API report, it is easier to use a Report API `mlreportgen.report.ListOfFigures`, `mlreportgen.report.ListOfTables`, or `mlreportgen.report.ListOfCaptions` object.

### See Also

`mlreportgen.report.ListOfFigures` | `mlreportgen.report.ListOfTables` |  
`mlreportgen.report.ListOfCaptions` | `mlreportgen.dom.LOF` | `mlreportgen.dom.LOT` |  
`mlreportgen.dom.LOC`

### More About

- “Create a List of Figures or Tables in a Report API Report” on page 3-40
- “Create a List of the Captions or Titles of Related Report Elements in a Report API Report” on page 3-46
- “Update a Table of Contents or Generated List in a Word Document” on page 3-37
- “Automatically Number Document Content” on page 13-101

## Create a List of the Captions or Titles of Related Report Elements in a Report API Report

In a Report API report, you can create a list of the captions or titles of related report elements, such as equations, so that you can browse the list and easily navigate to a particular report element. To create a list of report elements, use an `mlreportgen.report.ListOfCaptions` reporter.

---

**Note** For a list of figures, use an `mlreportgen.report.ListOfFigures` reporter. For a list of tables, use an `mlreportgen.report.ListOfTables` reporter. See [Create a List of Figures or Tables](#).

---

To use a `ListOfCaptions` reporter, you must create a paragraph for the caption or title of the report element. Then, associate the paragraph with an automatic numbering stream name that has a name that matches the automatic numbering stream name of the `ListOfCaptions` object.

A `ListOfCaptions` reporter creates a placeholder for a list in a new section of the report. Typically, you add a list of captions section after the table of contents and before the sections that contain the report content. In a PDF or Microsoft Word reports, the list of captions is located where you put it in the report. In an HTML reports, the section is located in a sidebar.

The way the list of captions is generated depends on the report type:

- PDF — The Report API generates the list during report generation.
- Word — You must generate the list by updating the Word document in your report generation program or interactively in Word. See “Update a Table of Contents or Generated List in a Word Document” on page 3-37.
- HTML — The HTML browser generates the list.

Each list item in a list of captions links to a caption or title in the report. In PDF and Word reports, a list item also includes the page number of the caption or title in the report and a leader fills the space between the caption or title and the page number.

To create a list of captions:

- 1** Create a Report API report.
- 2** Create an `mlreportgen.report.ListOfCaptions` object.
- 3** Specify the title of the list of captions section by setting the `Title` property of the `ListofCaptions` object.
- 4** Choose a numbering stream name, for example, `equation`. Set the `AutoNumberStreamName` property of the `ListofCaptions` object to the numbering stream name.
- 5** Add the `ListofCaptions` object to the report.
- 6** Create chapters or sections for the report content.
- 7** Add content to the chapters or sections. Create `mlreportgen.dom.Paragraph` objects for the captions or titles of the report elements that you want to include in the list of captions. Use `mlreportgen.dom.AutoNumber` objects to associate the paragraphs with the same automatic numbering stream name that is assigned to the `AutoNumberStreamName` property of the `ListofCaptions` object.
- 8** Close the report.

## Create a Report That Has a List of Captions Section

This example shows how to create a list of captions section in a report. The example generates a report about physics equations and adds a list of the captions of the equations to the report.

Import the DOM and Report API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
import mlreportgen.report.*
```

Create a report.

```
rpt = Report("physics", "docx");
```

Add a title page and a table of contents to the report.

```
tp = TitlePage();
tp.Title = "Physics Principles";
tp.Author = "John Doe";
append(rpt, TableOfContents);
```

Create a list of captions section.

```
loc = ListOfCaptions();
loc.Title = "List of Equations";
```

Define an automatic numbering stream name to use with the ListOfCaptions object.

```
equationStreamName = "equation";
loc.AutoNumberStreamName = equationStreamName;
```

Add the list of captions section to the report.

```
append(rpt, loc);
```

Create chapters for the report content. For this example, create two chapters. Each chapter has one equation.

Create the first chapter. Add a paragraph and add an equation to it.

```
ch1 = Chapter("Force");
append(ch1, Paragraph("Calculate force by multiplying mass and acceleration."));
eq1 = Equation("F = ma");
append(ch1, eq1);
```

Create a paragraph for the equation caption and associate the automatic numbering stream name with the caption.

```
caption1 = Paragraph("Equation ");
append(caption1, AutoNumber(equationStreamName));
```

Define a style for the captions. Include an mlreportgen.report.CounterInc object to increment the counter for the numbering stream.

```
equationCaptionsStyle = {HAlign("center"), CounterInc(equationStreamName), WhiteSpace("preserve")};
caption1.Style = equationCaptionsStyle;
```

Add the rest of the caption text and add the caption to the chapter.

```
append(caption1, ".");  
append(caption1, " Force");  
append(ch1, caption1);
```

Add the chapter to the report.

```
append(rpt, ch1);
```

Create the second chapter. Add a paragraph and add an equation to the paragraph.

```
ch2 = Chapter("Momentum");  
append(ch2, Paragraph("Calculate momentum by multiplying mass and velocity."));  
eq2 = Equation("p = mv");  
append(ch2, eq2);
```

Create a paragraph for the equation caption and associate the automatic numbering stream name with the caption.

```
caption2 = Paragraph("Equation ");  
append(caption2, AutoNumber(equationStreamName));
```

Specify the style for the caption.

```
caption2.Style = equationCaptionsStyle;
```

Add the rest of the caption text and add the caption to the chapter.

```
append(caption2, ".");  
append(caption2, " Momentum");  
append(ch2, caption2);
```

Add the chapter to the report.

```
append(rpt, ch2);
```

Close and view the report.

```
close(rpt);  
rptview(rpt);
```

Here is the list of equations in the report:

## List of Equations

Equation 1. Force .....	1
Equation 2. Momentum.....	2

### Customize a List of Captions Section Title

You can specify the title of a list of captions section by setting the `Title` property of the `ListOfCaptions` reporter. To customize the style of the title, such as the font family, color, or size, use one of these approaches:

- Specify the title as an `mlreportgen.dom.Text` object and specify the style using the object properties.
- Specify the title style in the template used by the `ListOfCaptions` reporter.
- Specify the title style in the template used by the title reporter for the list of captions section title. See “Use a Custom Template for a List of Captions Title” on page 11-238.

## See Also

`mlreportgen.report.ListOfFigures` | `mlreportgen.report.ListOfTables` |  
`mlreportgen.report.ListOfCaptions`

## More About

- `mlreportgen.dom.CounterInc`
- `mlreportgen.dom.AutoNumber`
- “Create a List of Figures or Tables in a Report API Report” on page 3-40
- “Update a Table of Contents or Generated List in a Word Document” on page 3-37
- “Create a List of Figures, Tables, or Other Report Elements in a DOM API Report” on page 3-45
- “Automatically Number Document Content” on page 13-101



# Set Up a Report

---

- “Report Setups” on page 4-2
- “Create a New Setup File” on page 4-4
- “Open a Report Setup” on page 4-5
- “Close a Report Setup” on page 4-7
- “Save a Report Setup” on page 4-8
- “Load Report Setup into MATLAB Workspace” on page 4-9
- “Insert Components” on page 4-10
- “Set Component Properties” on page 4-11
- “Move Components” on page 4-12
- “Delete Components” on page 4-13
- “Deactivate Components” on page 4-14
- “Send Components to the MATLAB Workspace” on page 4-15

## Report Setups

<b>In this section...</b>
---------------------------

“Setup Hierarchy” on page 4-2
-------------------------------

“Setup Files” on page 4-2
---------------------------

“Create a Report Setup” on page 4-2
-------------------------------------

A report setup is a set of MATLAB objects, called components, that specified the content and form of a report.

The MATLAB Report Generator provides a setup editor, called the Report Explorer, that you used to create and edit report setups.

Once you created a setup, you can generate a report from it, using the Report Explorer or MATLAB commands.

### Setup Hierarchy

A report setup has a hierarchical structure that generally mirrors the structure of the type of report that it defines.

For example, a report typically has a title page and one or more chapters. Each chapter contains one or more sections, each of which contains one or more paragraphs, figures, tables, lists, etc. A report setup typically comprises components that correspond to these structural elements of a report.

In a report setup, child-parent relationships among the components correspond to the containment relationships among the structural elements of the report. In particular, all setups contain a root component that serves as the ancestor for all other components in the setup. The root component also specifies the setup name and report generation options, such as the document type of the generated report (for example, HTML or PDF) and the path for the generated report. The root component typically parents a title page component and one or more chapter components that in turn parent one or more section components that parent one or more paragraph, figure, table, and list components.

### Setup Files

The report generator stores setups in files called setup files. The name of a setup file consists of the name of the setup that it stores followed by the file extension `.rpt`. For example, the name of the setup file for a setup named `myreport` would be `myreport.rpt`.

### Create a Report Setup

To create a report setup:

- Create a new setup file on page 4-4.
- Insert components on page 4-10 to define the content and format of the report.
- Set component properties on page 4-11.
- Save the setup on page 4-8.



Once you create a template, you can execute it to generate an instance of the type of report that it defines.

## Create a New Setup File

In this section...
“Create Setup File Using the Report Explorer” on page 4-4
“Create Setup File Programmatically” on page 4-4
“Working with Setup Files” on page 4-4
“Report Description” on page 4-4

Create a new setup file either interactively from the Report Explorer or programmatically.

### Create Setup File Using the Report Explorer

- 1 If the Report Explorer is not already open, from the MATLAB Toolstrip, in the **Apps** tab, in the **Database Connectivity and Reporting** section, click **Report Generator**.
- 2 In the Report Explorer, use *one* of these approaches:
  - Select **File > New**.
  - On the Report Explorer toolbar, click the new template button.

The Outline pane displays a new setup named Unnamed, as a child of the Report Generator node.

### Create Setup File Programmatically

To create a setup file programmatically (from the MATLAB command line), use the `setedit` command. For example, assuming a setup named `myreport` does not already exist in the current directory, use the following command:

```
setedit myreport.rpt
```

### Working with Setup Files

For details about performing operations on report setup files, see:

- “Open a Report Setup” on page 4-5
- “Close a Report Setup” on page 4-7
- “Save a Report Setup” on page 4-8

### Report Description

To record notes and comments about your report setup, use the **Report Description** field. This text that you enter appears in the Properties pane when you select a report setup file in the Outline pane.

## Open a Report Setup

### In this section...

“Opening a Setup on the MATLAB Path” on page 4-5

“Opening a Setup Not on the MATLAB Path” on page 4-5

“Opening a Setup Programmatically” on page 4-6

To make changes to a saved report setup, you must open its setup file. Open a report setup either interactively from the Report Explorer or programmatically.

### Opening a Setup on the MATLAB Path

---

**Tip** Use the `setedit` command to obtain a list of all the report setups on the MATLAB path.

---

To open a setup that resides on the MATLAB path:

- 1 If the Report Explorer is not already open, from the MATLAB Toolstrip, in the **Apps** tab, in the **Database Connectivity and Reporting** section, click **Report Generator**.
- 2 In the Report Explorer, in the Outline pane on the left, select the Report Generator node.

The Library pane in the middle displays a list of all the setup files that exist on the MATLAB path.

- 3 In the Library pane, select the setup file that you want to open.

The setup properties dialog box appears in the Properties pane on the right.

- 4 To open the setup, in the Report Explorer use *one* of these approaches:
  - On the Properties pane, click the **Open report** button.
  - On the Library pane, double-click the entry for the setup.
  - On the Library pane, from the context menu for the setup, select **Open report**.

The setup appears in the Outline pane as a child of the Report Generator node.

### Opening a Setup Not on the MATLAB Path

---

**Tip** Use the `setedit` command to obtain a list of all the report setups on the MATLAB path.

---

To open a setup that resides off the MATLAB path:

- 1 If the Report Explorer is not already open, from the MATLAB Toolstrip, in the **Apps** tab, in the **Database Connectivity and Reporting** section, click **Report Generator**.
- 2 In the Report Explorer, select **File > Open** or select the file open button on the Report Explorer toolbar.

A file browser opens.

- 3 Use the file browser to find the report setup in your file system and enter the setup name in the file browser **File name** field.

- 4 Select the file browser **Open** button.

The setup appears in the Outline pane as a child of the Report Generator node.

### Opening a Setup Programmatically

To open a report programmatically, use the `setedit` command. For example, the following command opens the `simple-report.rpt` example that comes with the MATLAB Report Generator.

```
setedit simple-report
```

This command opens the Report Explorer, if it is not already open, and opens the `simple-rpt` setup in the Report Explorer.

---

**Tip** If a setup exists on the MATLAB path, you do not need to specify its full path when using the `setedit` command. Use the `setedit` command to obtain a list of all the report setups on the MATLAB path.

---

The newly opened report appears in the Outline pane as a child of the Report Generator node.

## Close a Report Setup

<b>In this section...</b>
“Close a Setup Using the Report Explorer” on page 4-7
“Close a Setup Programmatically” on page 4-7

Closing a setup removes the setup from the Report Explorer and from memory.

### Close a Setup Using the Report Explorer

- 1 In the Report Explorer, in the Outline pane, select the setup root node.
- 2 In Report Explorer, use *one* of these approaches:
  - Click the **Delete** button.
  - Select **File > Close**.
  - From the context menu of the root node of the setup file, select **Close**.

### Close a Setup Programmatically

You can close a report that you have previously opened. For example, the following code opens a setup and then closes it.

```
setup('simple-report.rpt');  
root = RptgenML.Root;  
root.closeReport('simple-report');
```

## Save a Report Setup

<b>In this section...</b>
“Save a Setup Under Its Existing Name” on page 4-8
“Save a Setup Under a New Name” on page 4-8

### Save a Setup Under Its Existing Name

- 1 In the Report Explorer, in the Outline pane, select the setup root node.
- 2 In Report Explorer, use *one* of these approaches:
  - Click the **Save** button.
  - Select **File > Save**.
  - From the context menu of the root node of the setup file, select **Save**.

### Save a Setup Under a New Name

- 1 In the Report Explorer, in the Outline pane, select the setup root node.
- 2 Select **File > Save As**.

A file browser opens.
- 3 Use the file browser to select a new path for the setup.
- 4 In the file browser, click **Save**.

## Load Report Setup into MATLAB Workspace

To load a setup into the MATLAB workspace without loading it into the Report Explorer, use the `rptgen.loadRpt` function.

You can then modify the setup programmatically. For example, the following code loads a setup into memory, sets its output type to PDF, and generates a report.

```
setupRoot = rptgen.loadRpt('simple-report');  
setupRoot.Format = 'pdf';  
setupRoot.execute;
```

## Insert Components

In this section...
"Point-and-Click Method" on page 4-10
"Drag-and-Drop Method" on page 4-10
"Fix Context Violations" on page 4-10

### Point-and-Click Method

- 1 In the Report Explorer, in the Outline pane, select the parent node of the component to be inserted. For example, if you are inserting a paragraph into a section, select the section that will contain the paragraph.
- 2 In the Library pane, select the type of component that you want to insert in the report setup.
- 3 In the Properties pane, select the **Add component to current report** button.

### Drag-and-Drop Method

- 1 In the Report Explorer, in the Library pane, select the type of component that you want to insert in the setup.
- 2 Drag the component from the Library pane into the Outline pane and drop it onto the parent of the component to be created.

### Fix Context Violations

The Report Explorer allows you to insert components into invalid contexts.

For example, a **Chapter/Subsection** component is a valid parent for a **Paragraph** component, but not vice-versa. Nevertheless, the Report Explorer allows you to insert a **Chapter/Subsection** as a child of a Paragraph. If you insert a component in an invalid context, the Report Explorer displays a warning.

Although you can create an invalid setup hierarchy, you cannot generate a report from an invalid hierarchy. You must fix the context violations first. For example, move components from invalid contexts to valid contexts (see "Move Components" on page 4-12).



## Set Component Properties

In this section...
“Edit Component Property Values” on page 4-11
“Computed Property Values” on page 4-11

### Edit Component Property Values

Most components have properties that you can set to select optional features. For example, the **Text** component lets you specify the color of the text that it generates among other properties.

To set component properties:

- 1 In the Report Explorer, in the Outline pane, select the component.

The Properties dialog box for the component appears in the Properties pane.

- 2 Use the Properties dialog box to set component properties.

### Computed Property Values

During report generation, the Report Generator can compute the values of component properties, using MATLAB expressions that you specify. This enables dynamic creation of report content. For example, you can use MATLAB expressions to compute the content of **Paragraph** components and the value of looping components that generate repeated content.

You can use MATLAB expressions to compute the value of any character vector property of a component. To specify a MATLAB expression as a character vector property value, in the Properties dialog box, in the property edit box, enter %<expr>, where `expr` is a MATLAB expression that evaluates to a character vector.

## Move Components

### In this section...

“Point-and-Click Method” on page 4-12

“Drag-and-Drop Method” on page 4-12

### Point-and-Click Method

- 1 In the Report Explorer, in the Outline pane, select the component that you want to move.
- 2 Reposition the component in the setup hierarchy, using *one* of these approaches:
  - On the Report Explorer toolbar, use the move buttons.
  - From the **Edit** menu, use the move commands.
  - From the context menu of the component, use the move commands.

---

**Note** The move buttons and commands are enabled only if they are valid in the context of the component to be moved. For example, if a component cannot move further to the right in the hierarchy, the **Move Right** button is disabled.

---

The following table summarizes the available move buttons and commands.

Move Command or Button	Effect
<b>Move Up</b>	Moves a component ahead of the sibling that formerly preceded it in the hierarchy. If the component is the first child of its parent, the component becomes a sibling of its former parent.
<b>Move Down</b>	Moves a component after the sibling that formerly followed it in the hierarchy. If a component is the last sibling of its parent, it moves up one level in the hierarchy to become a sibling of its former parent.
<b>Move Left</b>	Moves a component up one level in the hierarchy. The component becomes a sibling of its former parent.
<b>Move Right</b>	Moves a component down one level in the hierarchy. The component becomes the child of the sibling that formerly preceded it in the hierarchy.

### Drag-and-Drop Method

- 1 In the Report Explorer, in the Outline pane, select the component that you want to move.
- 2 Drag the component and drop it on the component that you want to be its parent.

## Delete Components

- 1 In the Report Explorer, in the Outline pane, select the component that you want to delete.
- 2 Delete the component, using *one* of these approaches:
  - On the Report Explorer toolbar, click the **Delete** button.
  - Select **Edit > Delete**.
  - From the context menu of the component, select **Delete**.

# Deactivate Components

You can deactivate any component in a report setup. Deactivating a component causes it to be skipped during generation of a report.

Deactivating components can be useful for debugging setups. For example, you can deactivate a component that you suspect is causing an error or you can activate only the components you want to debug, thereby cutting the time required to verify a fix.

To deactivate (or reactivate) a component:

- 1 In the Report Explorer, in the Outline pane, select the component that you want to deactivate (or reactivate).
- 2 Select the appropriate **Activate/Deactivate Component** option from either the **Edit** menu or from the context menu of the component.

## Send Components to the MATLAB Workspace

You can send the components of a setup from the Report Explorer to the MATLAB workspace. This allows you to inspect and set their properties at the MATLAB command line.

Sending components to the workspace can be useful for creating or debugging MATLAB programs that create report setups and generate reports from them.

To send a component to the MATLAB workspace:

- 1** In the Report Explorer, in the Outline pane, select the component that you want to send to the workspace.
- 2** From the context menu of the component, select **Send to Workspace**.



# Generate a Report

---


- “Generate a Report” on page 5-2
- “Select Report Generation Options” on page 5-3
- “Report Generation Preferences” on page 5-9
- “Change Report Locale” on page 5-12
- “Convert XML Documents to Different File Formats” on page 5-13
- “Create a Report Log File” on page 5-16
- “Generate MATLAB Code from Report Setup File” on page 5-17
- “Troubleshooting Report Generation Issues” on page 5-19

## Generate a Report

In this section...
“Run a Report” on page 5-2
“Report Output Options” on page 5-2

### Run a Report

You can generate a MATLAB Report Generator report using one of these methods:

- In the Report Explorer Outline pane, select a report and do one of these actions:
  - In the Report Explorer toolbar, click the Report button .
  - Press **CTRL+R**.
  - Select **File > Report**.
- From the MATLAB command line, enter the `report` command. For example, to print the `system1_description` report in PDF format, enter:

```
report system1_description -fpdf
```

### Report Output Options

Before you generate a report, you can set options to control aspects of report generation processing such as:

- Output file format (HTML, Microsoft Word, or PDF)
- Style sheet or templates for the selected output file format, to control the layout of the report
- Output file location
- Whether to view the report after it is generated

For details, see:

- “Report Output Format” on page 5-4
- “Location of Report Output File” on page 5-3
- “Create a Report Log File” on page 5-16
- “Report Description” on page 4-4
- “Change Report Locale” on page 5-12



## Select Report Generation Options

In this section...
"Report Options Dialog Box" on page 5-3
"Location of Report Output File" on page 5-3
"Report Output Format" on page 5-4
"PDF Style Sheets" on page 5-6
"Web Style Sheets" on page 5-6
"RTF (DSSSL Print) and Word Style Sheets" on page 5-7
"Report Generation Processing" on page 5-7

### Report Options Dialog Box

To specify report generation options for a report, in the Report Explorer, use the Report Options dialog box. The Report Options dialog box appears when you select the report from the outline view.

To set the defaults for these options, use the Report Generator preferences. For details, see "Report Generation Preferences" on page 5-9.

If you are creating a form-based report, instead use the Report Form Options dialog box. See "Report Form" on page 16-13.

### Location of Report Output File

Choose a folder to store the report file. You must have write privileges for that folder.

#### Folder

In the Report Explorer, in the Report Options dialog box, use the **Directory** field to specify the name of the folder in which to store the generated report file. Specify a folder to which you have write privileges.

The following table summarizes the report file location options.

Folder	Option
The same folder as the report setup file	Same as setup file
The current working folder	Present working directory
Temporary folder	Temporary directory
Another folder	Custom. Use the <b>Browse</b> button (...) to select from a list of directories.

You can use %<VariableName> notation to specify a folder in the **Custom** text box.

#### Report File Name

In the Report Explorer, in the Report Options dialog box, use the **Filename** field to specify a file name for the report file. Select one of the following options.

File Name	Option
The same file name as the report setup file	Same as setup file (default)
A file name different from the report setup file name	Custom. Enter the name of the report.

You can use %<VariableName> notation to specify a file name in the **Custom** text box.

### Increment to Prevent Overwriting

To maintain the previous version of the setup file when you save updates to the setup file, select **If report already exists, increment to prevent overwriting**.

### Image Output File Location

Images are placed in a folder with the same name as the report file. For example, `testreport.html` images are placed in a folder named `testreport_files`.

## Report Output Format

Under **Report Output Type and Style Sheets**, from the **File format** list, select the report output format. You can generate reports whose formatting is based on templates or based on Report Explorer style sheets.

### File Formats Using Report Templates

For faster report generation, set **File format** to use a template. Select one of these options:

- Direct PDF (from template)
- HTML (from template) — Select this option if you want your report's table of contents to expand and collapse.
- PDF (from Word template)
- Single-file HTML (from template) — Select this option if you want your report's table of contents to expand and collapse and want the HTML report as a single file.
- Word (from template)

If you select a `from template` output format, then you can use a default template or a customized template. For more information about using templates for report generation, see "Generate a Report Using a Template" on page 7-10. For information on customizing templates, see "Customize Report Conversion Templates".

### File Format Using Report Explorer Style Sheets

The output formats that do not include `from template` in the name use Report Explorer style sheets for formatting. For those output formats, select a style sheet from the **Style Sheet** list.

Viewer	Format	Description	Style Sheets
Adobe® Acrobat® Reader	Adobe Acrobat (PDF)	Produce a PDF that you can view using Adobe Acrobat Reader software.  See “PDF: Image Formats” on page 5-5.	PDF (see “PDF Style Sheets” on page 5-6)
Word processor	Word Document (RTF) or Rich Text Format	Produce output that is compatible with most word-processing packages, including Microsoft Word software.  See “RTF: Display of Hidden Content” on page 5-5.	Print (see “RTF (DSSSL Print) and Word Style Sheets” on page 5-7)
DocBook	DocBook (no transform)	Produce a report in DocBook format	N/A

---

**Tip** To create and use customized styles, see “Create a New Style Sheet” on page 9-4.

---

### PDF: Image Formats

PDF reports support only bitmap (.bmp), JPEG (.jpg), and Scalable Vector Graphics (.svg).

The SVG format is supported only for Simulink models and Stateflow® charts. For example, MATLAB figures do not display in SVG when you select the SVG format for PDF reports.

### RTF: Display of Hidden Content

RTF reports use placeholders (field codes) for dynamically generated content, such as page numbers or images.

On Windows platforms, to display that content, press **Ctrl+A**, and then press **F9**.

On Linux® and Mac platforms, use the field code update interface for the program that you are using to view the RTF document.

### Change the Default Output Format

In the **Report Generator Preferences** pane, use the **Format ID** preference to specify the default output format for reports.

### Style Sheets

For each output format, you can choose from several style sheets for each report output format. For details, see:

- “PDF Style Sheets” on page 5-6
- “Web Style Sheets” on page 5-6
- “RTF (DSSSL Print) and Word Style Sheets” on page 5-7

---

**Note** Some Web and Print style sheets include a generated list of titles, which includes table titles and figures with titles.

---

## PDF Style Sheets

PDF Style Sheet	Description
Default print style sheet	Displays title page, table of contents, list of titles
Standard Print	Displays title page, table of contents, list of titles
Simple Print	Suppresses title page, table of contents, list of titles
Compact Simple Print	Minimizes page count, suppresses title, table of contents, list of titles
Large Type Print	Uses 12-point font (slightly larger than Standard Print)
Very Large Type Print	Uses 24-point font and landscape paper orientation
Compact Print	Minimizes white space to reduce page count
Unnumbered Chapters & Sections	Uses unnumbered chapters and sections
Numbered Chapters & Sections	Numbers chapters and sections
Paginated Sections	Prints sections with page breaks
Custom Header	Lets you specify custom headers and footers
Custom Titlepage	Lets you specify custom title page content and presentation
Verbose Print	Lets you specify advanced print options

## Web Style Sheets

Web Style Sheet	Description
Default HTML style sheet	HTML on a single page
Simulink book HTML style sheet	HTML on multiple pages; suppresses chapter headings and table of contents
Truth Table HTML style sheet	HTML on multiple pages; suppresses chapter headings and table of contents
Multi-page Web	HTML, with each chapter on a separate page
Single-page Web	HTML on a single page
Single-page Unnumbered Chapters & Sections	HTML on a single page; chapters and sections are not numbered
Single-page Numbered Chapters & Sections	HTML on a single page; chapters and sections are numbered

Web Style Sheet	Description
Single-page Simple	HTML on a single page; suppresses title page and table of contents
Multi-page Simple	HTML on multiple pages; suppresses title page and table of contents
Multi-page Unnumbered Chapters & Sections	HTML on multiple pages; chapters and sections are not numbered
Multi-page Numbered Chapters & Sections	HTML on multiple pages; chapters and sections are numbered

## RTF (DSSSL Print) and Word Style Sheets

RTF or Word Style sheet	Description
Standard Print	Displays title page, table of contents, list of titles
Simple Print	Suppresses title page, table of contents, list of titles
Compact Simple Print	Minimizes page count, suppresses title, table of contents, list of titles
Large Type Print	Uses 12-point font (slightly larger than Standard Print)
Very Large Type Print	Uses 24-point font and landscape paper orientation
Compact Print	Minimizes white space to reduce page count
Unnumbered Chapters & Sections	Uses unnumbered chapters and sections
Numbered Chapters & Sections	Numbers chapters and sections

## Report Generation Processing

The Report Options dialog box includes options for controlling report processing.

Option	Purpose
<b>View report after generation</b>	<p>When report generation finishes, the viewer associated with the report output format displays the report.</p> <p>On a Linux or Macintosh platform, the report generator calls the <code>soffice</code> command to open the document. On Linux, Apache OpenOffice™ or LibreOffice® must be installed. On Macintosh, Apache OpenOffice must be installed in the <code>/Applications</code> folder.</p> <p>To view the report manually, open the file from the location specified in the <b>Report Options</b> for the report, under <b>Report File Location</b>.</p>
<b>Auto save before generation</b>	Automatically save the report setup file before you generate a report.

<b>Option</b>	<b>Purpose</b>
<p><b>Compile model to report on compiled information</b></p>	<p>Ensure that a report reflects compiled values.</p> <p>By default, the Simulink Report Generator reports uncompiled values of Simulink parameters. The uncompiled values of some parameters, such as signal data types, can differ from the compiled values used during simulation.</p> <p>This option causes the report generator to compile a model before reporting on model parameters. After generating the report, the report generator returns the model to its uncompiled state.</p> <hr/> <p><b>Note</b> When you select this option, whenever report generation requires simulating the model (for example, the report includes a Model Simulation (Simulink Report Generator) component), the report generator uncompiles the model and then recompiles the model, if necessary, to report on model contents. If a report requires multiple compilations, the processing can be quite time-consuming.</p> <p>To minimize compilations, consider using separate reports to report on the contents of a model and on the results of simulating that model.</p>
<p><b>Evaluate this string after generation</b></p>	<p>Specify MATLAB code for processing to occur after the report is generated. For example, you could specify to close a model.</p>

## Report Generation Preferences

### In this section...

“Report Generator Preferences Pane” on page 5-9

“File Format and Extension” on page 5-10

“Image Formats” on page 5-10

“Report Viewing” on page 5-11

“Reset to Defaults” on page 5-11

“Edit HTML Command” on page 5-11

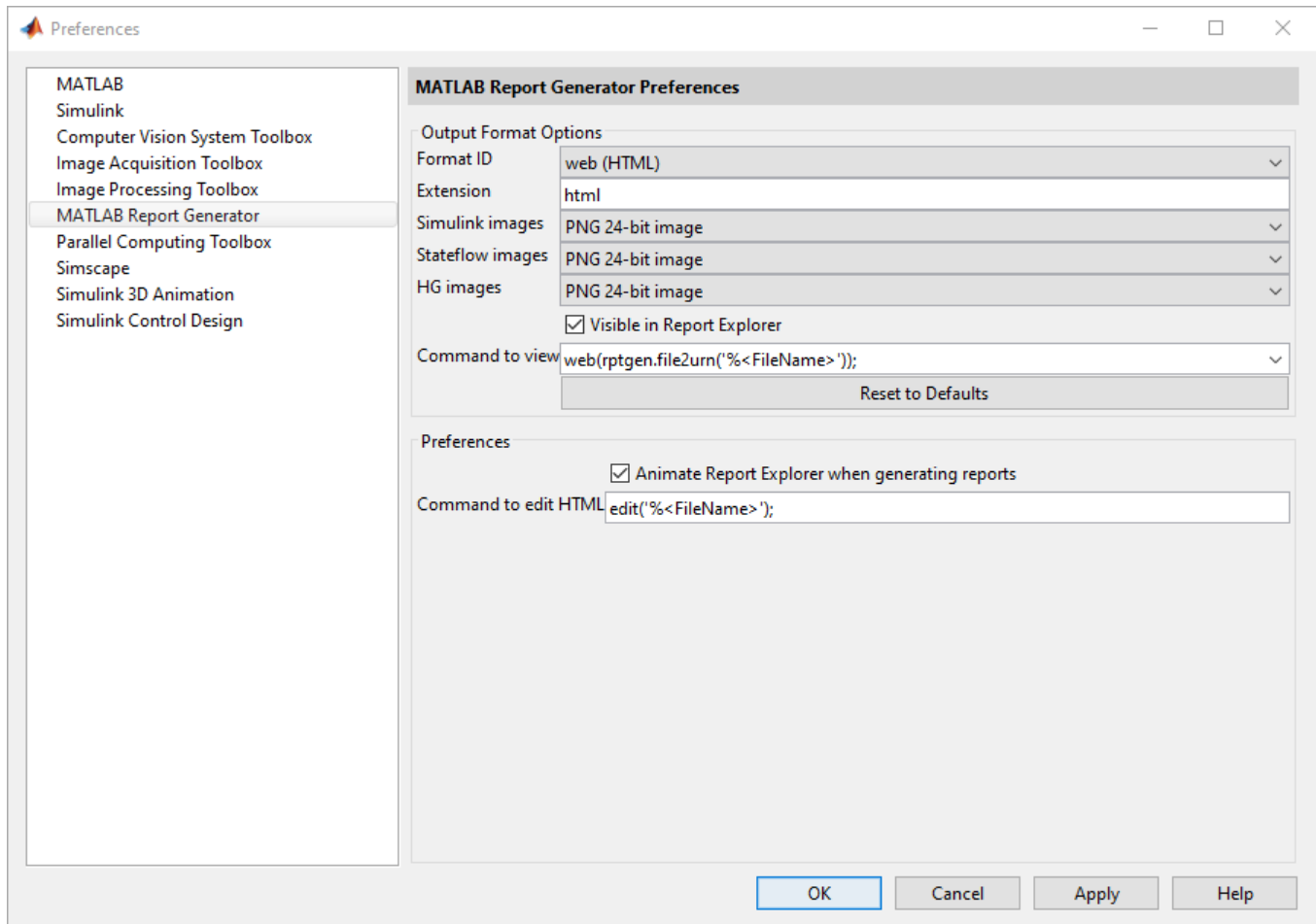
### Report Generator Preferences Pane

To set defaults for report generation options, use the Report Generator Preferences pane. You can override these preferences with the Report Options dialog box or with individual components.

To specify report generation options for a specific report, in the Report Explorer, use the Report Options dialog box. For details, see “Select Report Generation Options” on page 5-3.

To open the Report Generator Preferences pane, use one of these approaches:

- In the Report Explorer, select **File > Preferences**.
- From the MATLAB Toolstrip, in the **Home** tab, in the **Environment** section, select **Preferences > Report Generator**.



## File Format and Extension

To specify the default file format for reports, use the **Format ID** preference. The default preference is `web (HTML)`. You can select from a range of file formats, such as PDF or Microsoft Word.

If you have Adobe Acrobat and `acrobat.exe` is on your system path, you can set it as the default PDF format. Select a PDF format as the **Format ID**. Then, replace the command in **Command to view** with `system('acrobat %<FileName>');`

The **Extension** preference reflects the standard file extension for the file format specified with the **Format ID** preference. You can change the extension.

## Image Formats

To set the default image formats associated with the output format for a report, use the following preferences.

Preference	Purpose
<b>Simulink Images</b>	Specify the format for Simulink images to include in the report.



Preference	Purpose
<b>Stateflow Images</b>	Specify the format for Stateflow charts to include in the report.
<b>HG Images</b>	Specify the format for Handle Graphics® images to include in the report.

**Note** The default preferences for image formats should work in most viewing environments. However, some image formats do not display in some viewing environments.

Several components, such as the Figure Snapshot component, include an option for specifying the image format. The component setting overrides the image format preference.

## Report Viewing

To control how you view a generated report, you can set the following preferences.

Preference	Purpose
<b>View command</b>	Specify the MATLAB command you want to use to view the report.  Each file format has an associated default view command preference. You can modify the view command (for example, to support the use of a system browser).
<b>Animate Report Explorer when generating reports</b>	Select this check box if you want components in the Outline pane to be animated as the report generates. This box is selected by default.  To speed up the report generation processing, clear this preference.

## Reset to Defaults

To reset the Report Generator preferences under **Output Format Options**, click **Reset to Defaults**. Resetting to defaults does not affect the options under **Preferences**.

## Edit HTML Command

Enter the command to use to open HTML or PDF template files from the Report Explorer's Document Conversion Template Editor (see "Report Templates" on page 7-2). The default command opens the files in the MATLAB text editor.

To set this preference to an operating system command, use the MATLAB system command. Use the file name token %<FileName> to specify where in the command the template editor inserts the path to the file to edit. Make sure that the command is on your system path.

This example shows a command that opens Report Generator HTML-based template files in the notepad++ application. The ampersand character (&) executes the command in the background.

```
system('notepad++ %<FileName> &');
```

## Change Report Locale

Versions 2.0 and later of the MATLAB Report Generator and Simulink Report Generator software use the locale (system language settings) through the Oracle® Java interface; therefore, they should use the language specified on your system.

Alternatively, you can change the language directly in Java from the MATLAB command line. The following example sets the language to Italian:

```
java.util.Locale.setDefault(java.util.Locale.ITALY)
```

Alternatively, you can set the preferred language directly in your `.rpt` file:

- 1 Right-click the **Report** component and select **Send to Workspace**.

This displays the properties of the report, which are stored in the variable *ans*. Access the report's **Language** property from the command line through this variable. By default, **Language** is `auto`, which indicates that the system's default language is in use.

- 2 Override the default value of **Language** by setting this property to your desired language; for example, `en` for English or `it` for Italian.

## Convert XML Documents to Different File Formats

In this section...
“Why Convert XML Documents?” on page 5-13
“Convert XML Documents Using the Report Explorer” on page 5-13
“Convert XML Documents Using the Command Line” on page 5-14
“Edit XML Source Files” on page 5-15

### Why Convert XML Documents?

You can generate a report in a different output file format without regenerating it by using either the Report Explorer File Converter or the `rptconvert` command. These utilities convert DocBook XML source files created by the report-generation process into formatted documents such as HTML, RTF, or PDF.

---

**Note** The report-generation process can only convert XML source files created by the latest version of the software.

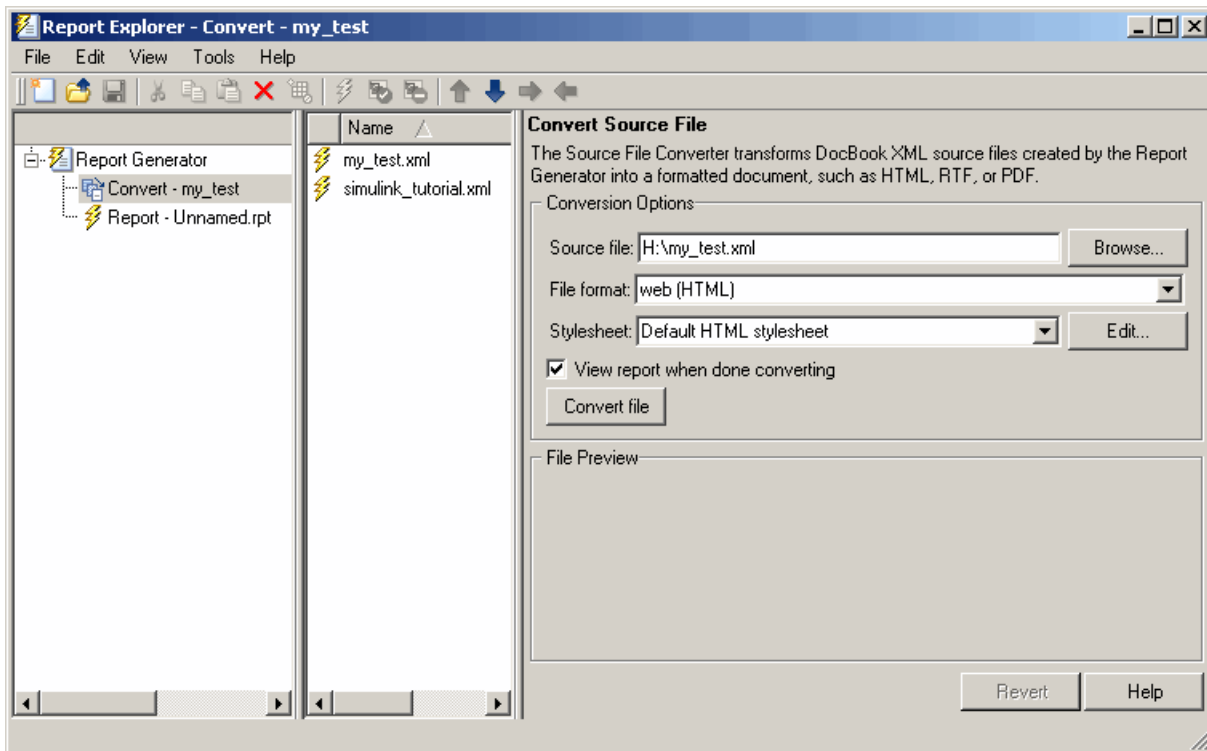
---

### Convert XML Documents Using the Report Explorer

To open the **Convert** Properties pane:

- 1 In the Report Explorer, select **Tools > Convert Source File**.

The Convert Source File Properties pane appears. All XML files in your current folder appear in the Options pane in the middle.



- 2 Select your XML source file using one of the following options:
  - Click **Browse** in the Properties pane on the right to browse to the location of your XML source.
  - Double-click a file name in the Options pane in the middle to automatically enter it into the **Source file** field in the Properties pane.
- 3 Select your output format and style sheet:
  - a In the **File format** text box, select an output format.
  - b In the **Style sheet** text box, select a style sheet. The style sheet choice depends on the specified output format. You can use a predefined or customized style sheet.

For more information about available formats and predefined style sheets, see “Report Output Format” on page 5-4.

For more information about customizing style sheets, see “Create a New Style Sheet” on page 9-4.
- 4 Use the **View Report when done converting** check box to indicate whether you want to view the report after it has conversion.
- 5 To begin the conversion, click **Convert file**.

## Convert XML Documents Using the Command Line

To convert files using the command line, use the rptconvertfunction.

## Edit XML Source Files

Before you send a source file to the converter, edit it as text in the Report Explorer:

- 1 In the Outline pane on the left, open the File Converter.
- 2 Right-click **MATLAB Report Generator** and select **Convert source file**.
- 3 In the Options pane in the middle, select the source file to edit.
- 4 In the Properties pane on the right, click **Edit as text**.
- 5 Use the MATLAB Editor to edit and save the text.

## Create a Report Log File

A log file describes the report setup file report-generation settings and components. A log file can be used for many purposes, including:

- As a debugger
- As a reference to a report setup file
- To share information about a report setup file through email

A log file includes the following information:

- Report setup file outline
- Components and their attributes
- Generation status messages currently displayed in the **Generation Status** tab

To generate a log file, click **File > Log File**. An HTML version of the log file with the name `<report_template_file_name_log>.html` is saved in the same folder as the report setup file.

## Generate MATLAB Code from Report Setup File

You can generate MATLAB code versions of report setup files in the form of a MATLAB file (\*.m). A MATLAB file of a report setup file is useful for various purposes, including generating reports and modifying report setup files programmatically.

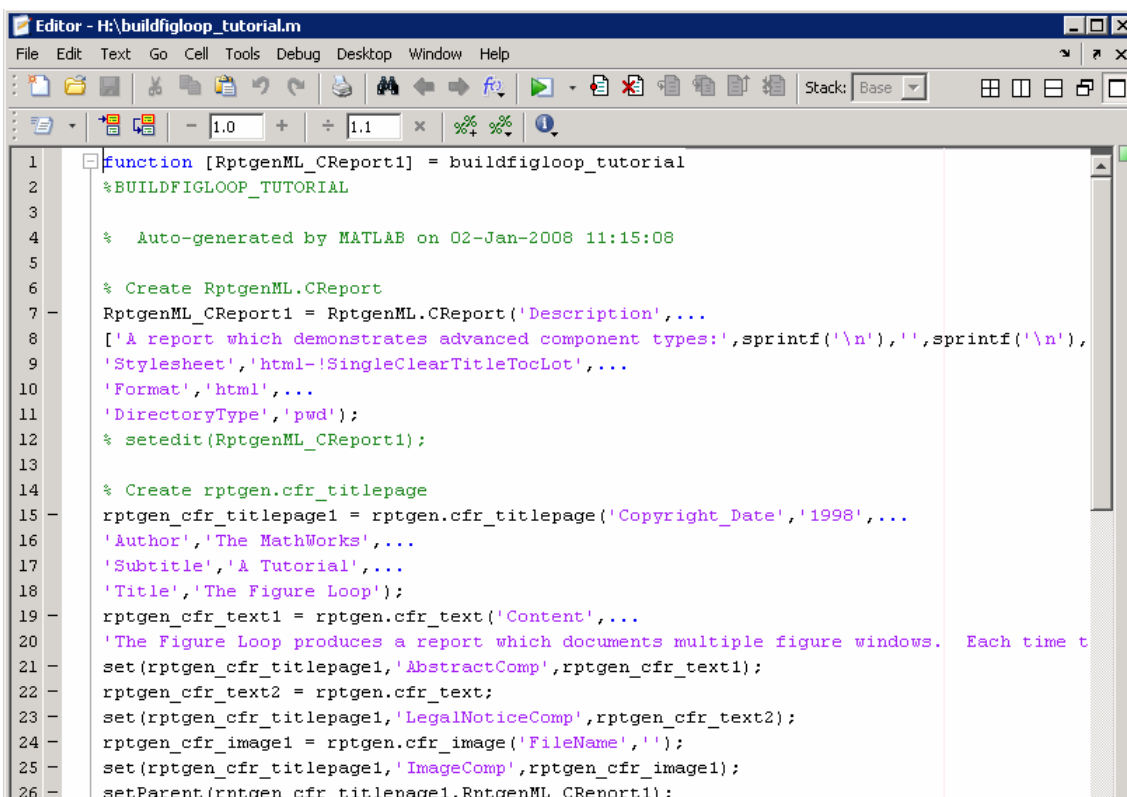
To update a MATLAB file, load a report setup file into the Report Explorer and click **File > Generate MATLAB File**. After the MATLAB file generates, it opens in the MATLAB Editor. The filename for the generated file is the file name of the report setup file, preceded by "build."

### Example 5.1. Generate Reports from MATLAB Files

This example generates a MATLAB file from the `figloop_tutorial.rpt` report setup file, which is part of the MATLAB Report Generator software. The example then uses the `report` function to generate a report from the MATLAB file. For more information about this function, see the `report` reference page.

- 1 Start the Report Explorer by entering `report` in the MATLAB Command Window.
- 2 In the Options pane in the middle, double-click `figloop_tutorial.rpt` to open its report setup file.
- 3 In the Outline pane on the left, click `Report - figloop_tutorial.rpt` to select it.
- 4 In the Report Explorer menu bar, click **File > Generate MATLAB File**.

The MATLAB Report Generator software generates MATLAB code for the `figloop_tutorial.rpt` report setup file. It saves this code in the `buildfigloop_tutorial.m` file in the folder you specify. Part of this file appears in the following figure.



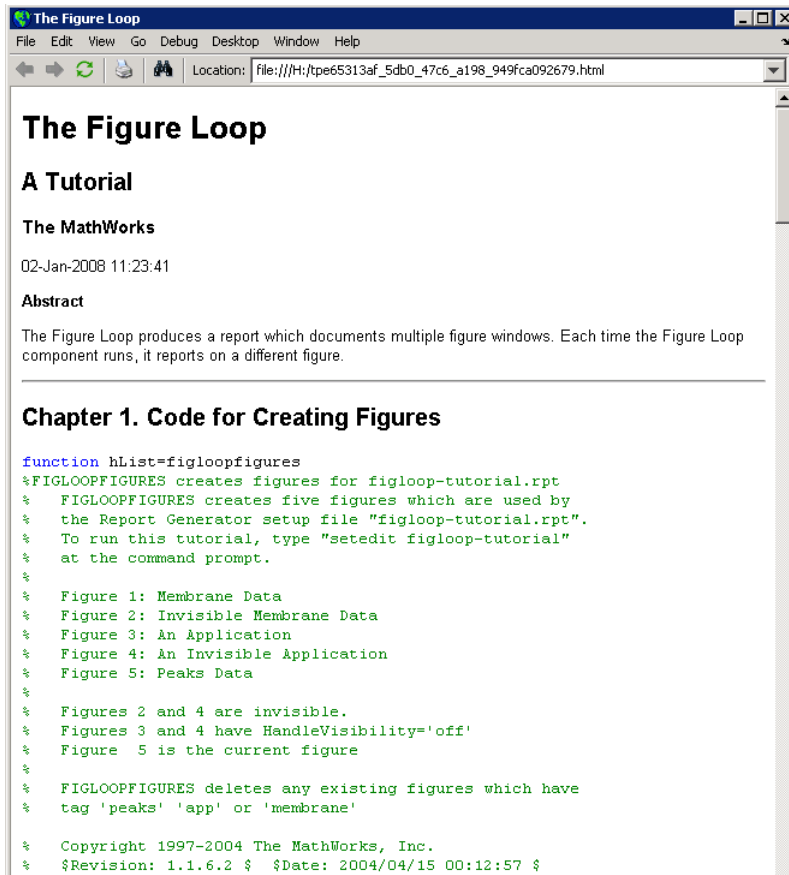
```

1 function [RptgenML_CReport1] = buildfigloop_tutorial
2     %BUILDFIGLOOP_TUTORIAL
3
4     % Auto-generated by MATLAB on 02-Jan-2008 11:15:08
5
6     % Create RptgenML.CReport
7     RptgenML_CReport1 = RptgenML.CReport('Description',...
8     ['A report which demonstrates advanced component types:',sprintf('\n'),' ',sprintf('\n'),
9     'Stylesheet','html-!SingleClearTitleTocLot',...
10    'Format','html',...
11    'DirectoryType','pwd');
12    % setedit(RptgenML_CReport1);
13
14    % Create rptgen.cfr_titlepage
15    rptgen_cfr_titlepage1 = rptgen.cfr_titlepage('Copyright_Date','1998',...
16    'Author','The MathWorks',...
17    'Subtitle','A Tutorial',...
18    'Title','The Figure Loop');
19    rptgen_cfr_text1 = rptgen.cfr_text('Content',...
20    'The Figure Loop produces a report which documents multiple figure windows. Each time t
21    set(rptgen_cfr_titlepage1,'AbstractComp',rptgen_cfr_text1);
22    rptgen_cfr_text2 = rptgen.cfr_text;
23    set(rptgen_cfr_titlepage1,'LegalNoticeComp',rptgen_cfr_text2);
24    rptgen_cfr_image1 = rptgen.cfr_image('FileName','');
25    set(rptgen_cfr_titlepage1,'ImageComp',rptgen_cfr_image1);
26    setParent(rptgen_cfr_titlepage1,RptgenML_CReport1);
  
```

- 5 To generate the `figloop_tutorial` report from this MATLAB file, run the following command in the MATLAB Command Window:

```
report(buildfigloop_tutorial);
```

The MATLAB Report Generator software runs and displays the report.





## Troubleshooting Report Generation Issues

### In this section...

“Memory Usage” on page 5-19

“HTML Report Display on UNIX Systems” on page 5-19

### Memory Usage

The Report Generator software has two converters for generating documents. One uses Java heap memory and the other does not.

To avoid Java heap memory issues, you can generate your report using the converter that does not use Java heap memory. To do so, under the **Report Options** for the report, set **File format** to one of the (from template) options, for example, HTML (from template).

If you select one of the other options, you are using the converter that uses Java heap memory and you might have memory issues. By default, MATLAB sets a limit of 384 MB on the amount of memory the Oracle Java Virtual Machine (JVM™) software can allocate. The memory that the report generation process uses to build a document must fit within this limit. If you are having trouble processing large reports, you can try increasing the amount of memory that the software can allocate by:

- Running MATLAB without a desktop
- Increasing the memory allocation limit

#### Run MATLAB Without a Desktop

To run the MATLAB software without a desktop, start MATLAB using the `-nodesktop` option. In this case, you must generate the report from the command line using the `report` command.

#### Increase the MATLAB JVM Memory Allocation Limit

To increase the amount of JVM memory available by increasing the MATLAB JVM memory allocation limit, from the MATLAB Toolstrip, in the **Home** tab, in the **Environment** section, click **Preferences**. Use the **General > Java Heap Memory** pane to increase the memory.

### HTML Report Display on UNIX Systems

HTML reports might not display in the Report Generator Web viewer on some UNIX® platforms. If this happens, configure the Report Generator software to launch an external browser.

- 1 In the Report Explorer, click **File > Preferences**.
- 2 Enter this command in the **View command** field, where *file name* is the name of your report setup file:

```
web(rptgen.file2urn('%file name'), '-browser')
```



# Add Content with Components

---

- “Components” on page 6-2
- “Report Structure Components” on page 6-4
- “Table Formatting Components” on page 6-5
- “Property Table Components” on page 6-6
- “Summary Table Components” on page 6-14
- “Logical and Looping Components” on page 6-17
- “Edit Figure Loop Components” on page 6-18

## Components

Components are MATLAB objects that specify the content of a report. Add components to specify the types of content that commonly occur in reports. The MATLAB Report Generator provides a set of components for specifying the types of content that commonly occur in MATLAB-based reports. The Simulink Report Generator provides additional components to facilitate generation of reports from Simulink models. You can also create custom components to handle content specific to your application.

Using the Report Explorer, you interactively combined components to create a report setup that specifies the content of a particular report or type of report. For general information about working with components, see:

- “Insert Components” on page 4-10
- “Set Component Properties” on page 4-11

You used a combination of the following types of components in your report setup file, based on your goals for the report.

Type of Component	Description
“Report Structure Components” on page 6-4	Include a title page, chapters, sections, paragraphs, lists, tables, and other standard document structure elements.
“Table Formatting Components” on page 6-5	Organize generated content into tables.
“Property Table Components” on page 6-6	Display tables with property name/property value pairs for objects.
“Summary Table Components” on page 6-14	Display tables with specified properties for objects.
“Logical and Looping Components” on page 6-17	Run child components a specified number of times. There are several looping components, including logical loops and Handle Graphics loops.

### Component Formatting

When you generate a report, in the Report Options dialog box, in the File format field you specify the type of report output you want. For example, you can generate a report in PDF, HTML, or Microsoft Word format.

For each format, you can choose to apply styles from either of these style definition sources:

- An HTML or Word report conversion template
- A Model Explorer style sheet for HTML, Word, or PDF.

The output format and the associated template or style sheet that you select for a report determines most aspects of the formatting of the generated report. For example, a report template or style sheet typically uses different font sizes for chapter titles and section titles. For details, see “Report Output Format” on page 5-4.

Several components include properties that you can set to specify formatting details for that specific instance of a component. For example, for the **MATLAB Property Table**, you can specify formatting such as whether to display table borders or the alignment of text in table cells.

## Report Structure Components

Use report structure components to organize the content of your report into chapters, sections, paragraphs, lists, tables, and other standard document structure elements. The table summarizes the report structure components.

Component	Usage
Title Page	Generate a title page for a report.
Chapter/Subsection	Parent components that generate the content of a chapter or chapter subsection.
Paragraph	Specify the content and text format of a paragraph of text. Can serve as the parent of one or more text components, enabling use of multiple text formats (for example, bold, regular, or italic) in the same paragraph.
Text	Format generated text.
List	Generate a list from a cell array of numbers or text or parent components (for example, Paragraph components) that specify the items in a list. You can create multilevel lists by embedding list components within list components.
Link	Generate a hyperlink from one location in a report to another or to an external location on the user's file system or the Worldwide Web.
Image	Insert an image into a report.
Array-Based Table	Generate a table from a cell array of numbers or text.
Table	Parent a table body component. See "Table Formatting Components" on page 6-5.

## Table Formatting Components

Use table formatting components to organize generated content into tables. The following table summarizes the table formatting components.

Component	Usage
Table	Parent a table body component. Can also parent column specification components and a table header and a table footer component. Specifies properties of the table as a whole (for example, its title, number of columns, and border).
Table Body	Parent the rows that make up the table body. Specifies the default vertical alignment of entries in a table body.
Table Column Specification	Specify attributes of a table column, such as its width and borders and the default horizontal alignment of column entries.
Table Entry	Parent a component that determines a table entry's content, such as a paragraph, image, list, or another table component. Specifies attributes of a table entry, such as the number of rows and columns that it spans.
Table Footer	Parent the row components that generate the content of a table footer.
Table Header	Parent the row components that generate the content of a table header.
Table Row	Parent the table entry components that generate the content of a table row.

**Tip** Inserting a Table component into a setup also inserts all the descendant components required to generate a 2x2 table, creating a table template. Edit this template to create a table that suits your needs.

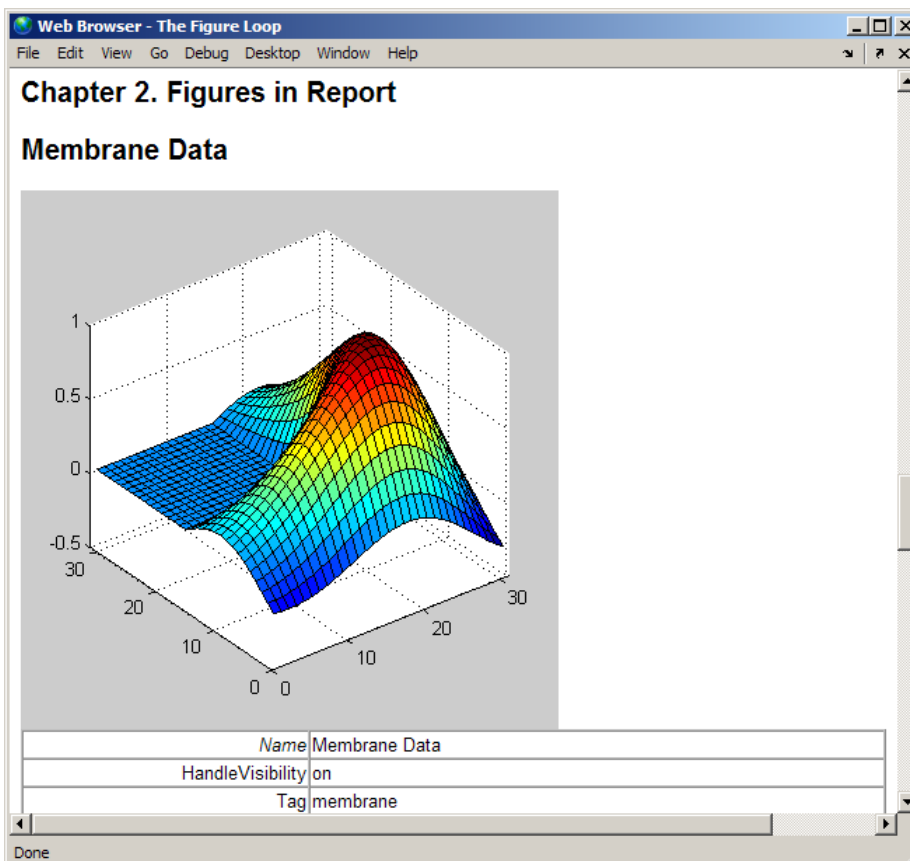
## Property Table Components

### In this section...

“About Property Table Components” on page 6-6  
 “Open the Example Report Template” on page 6-7  
 “Examine the Property Table Output” on page 6-7  
 “Select Object Types” on page 6-8  
 “Display Property Name/Property Value Pairs” on page 6-8  
 “Edit Table Titles” on page 6-10  
 “Enter Text into Table Cells” on page 6-10  
 “Add, Replace, and Delete Properties in Tables” on page 6-11  
 “Format Table Columns, Rows, and Cells” on page 6-12  
 “Zoom and Scroll” on page 6-13  
 “Select a Table” on page 6-13

### About Property Table Components

Property Table components display property name/property value pairs for objects in tables. The following example shows a property table from the `figloop-tutorial` report.





Many types of property table components are available, including:

- MATLAB Property Table
- Simulink Property Table (requires Simulink Report Generator)
- Stateflow Property Table (requires Simulink Report Generator)

The component used in this example represents MATLAB Report Generator property table components, all of which exhibit similar behavior.

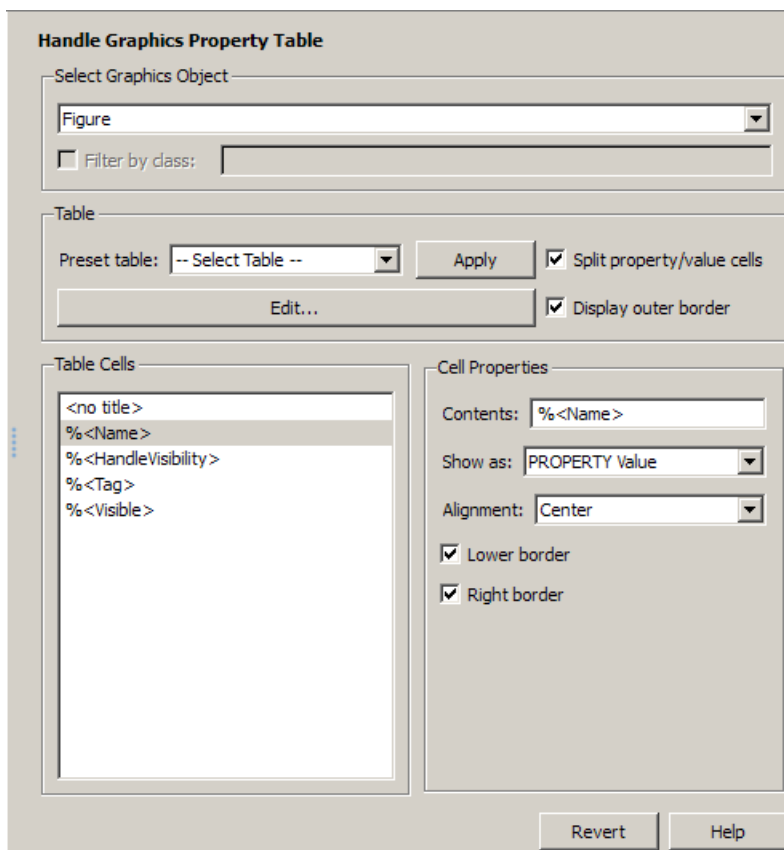
## Open the Example Report Template

This example uses the `figloop-tutorial` report template. To open the figure loop tutorial report template, at the MATLAB command line enter:

```
setedit figloop-tutorial
```

## Examine the Property Table Output

Property pages for all property table components are similar in form. In the Outline pane, select the **Figure Prop Table** component. To modify table settings, in the Handle Graphics Property Table dialog box, click the **Edit...** button.



## Select Object Types

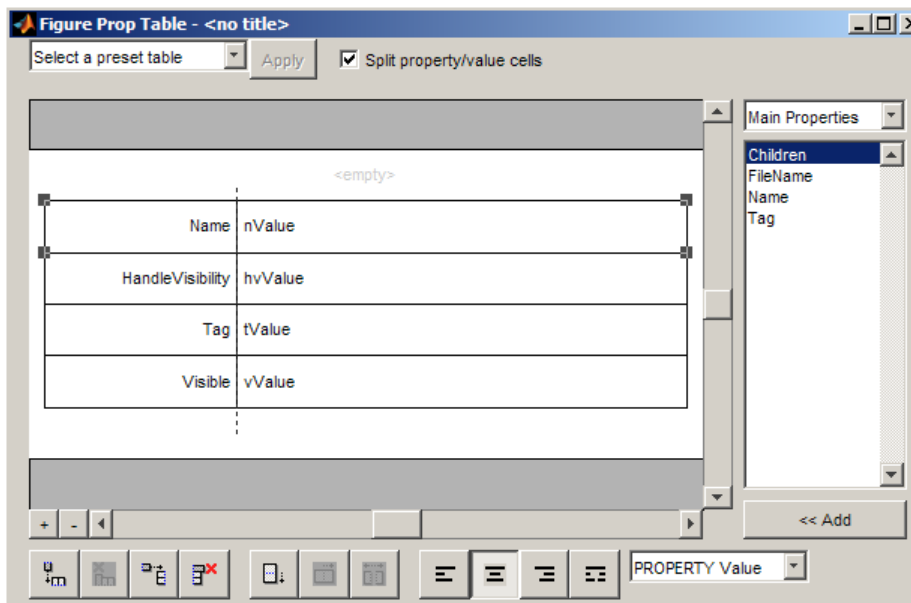
Property table components offer multiple object types on which to report. For example, the Handle Graphics Property Table lets you report on a figure, an axes object, or a Handle Graphics object.

You can select a different object type on which to report in the **Object type** list in the Properties pane for the component.

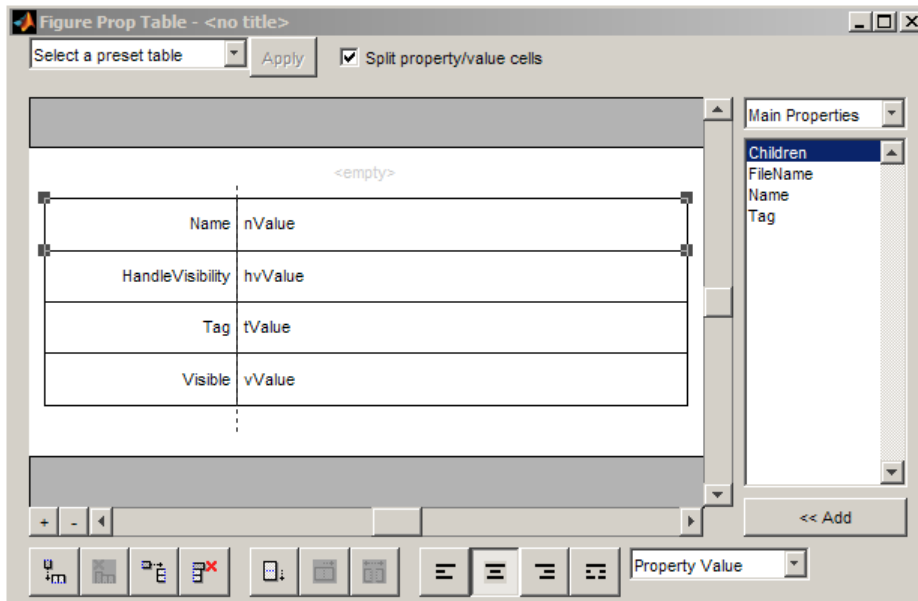
## Display Property Name/Property Value Pairs

### Split Property/Value Cells

- 1 In the Properties pane for the Handle Graphics Property Table component, clear the **Split property/value cells** check box.
- 2 Click **Edit**. The table is now in *nonsplit mode*. Nonsplit mode supports more than one property name/property value pair per cell and text.

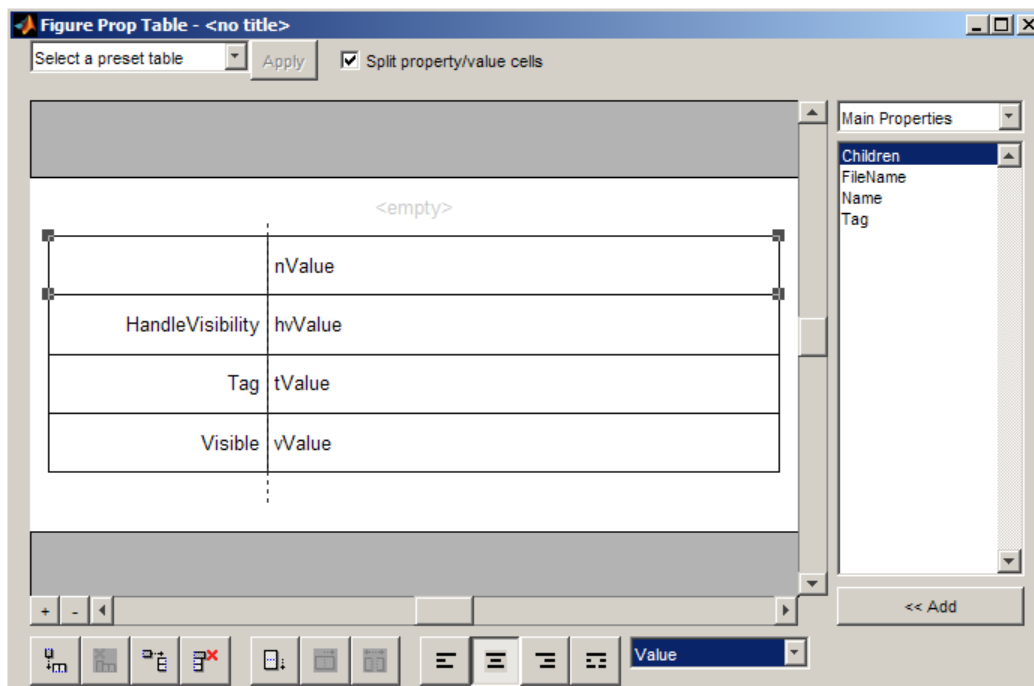


- 3 For the property name and property value to appear in adjacent horizontal cells in the table, select the **Split property/value cells** check box. The table is now in *split mode*. Split mode supports only one property name/property value pair per cell. If more than one property pair appears in a cell, only the first pair appears in the report; all subsequent pairs are ignored.



## Display Options

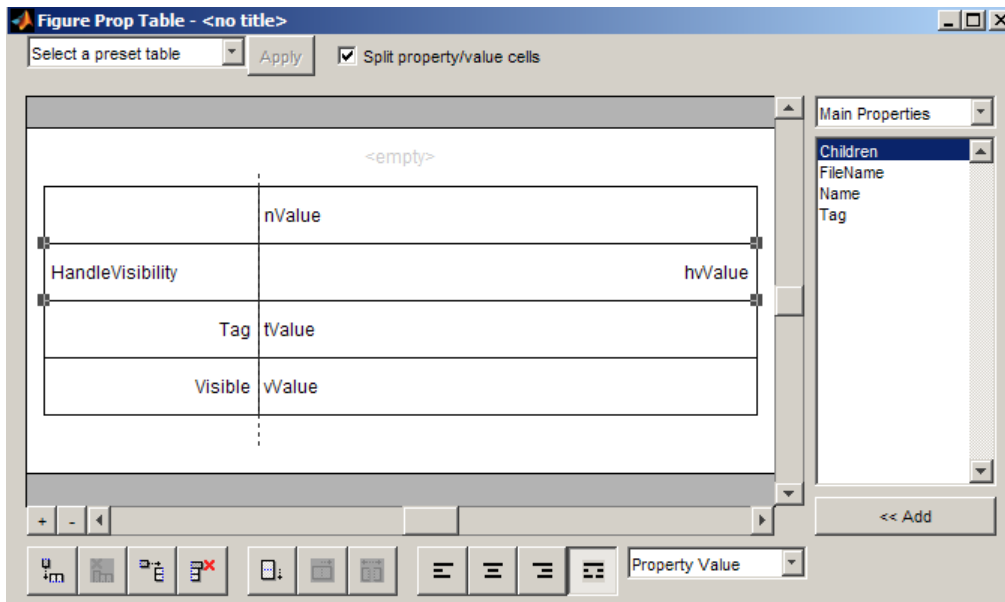
Property name/property value pairs can appear in cells in several ways. To specify how a given property name/property value pair appears in a cell, select that field in the table (for this tutorial, select the **Name** property). Choose **Value** from the display options drop-down list at the bottom of the dialog box. In the selected table row, only the value appears.



## Format Options

To specify alignment for text in a given cell, in the toolbar at the bottom of the dialog box use the four justification buttons.

Select the **HandleVisibility** table row. Then select the double-justify button (the last button to the right).



## Edit Table Titles

Table titles can contain properties and text. By default, the title of a table is the same as the value of the %<Name> property. You can modify this property to modify the table title.

---

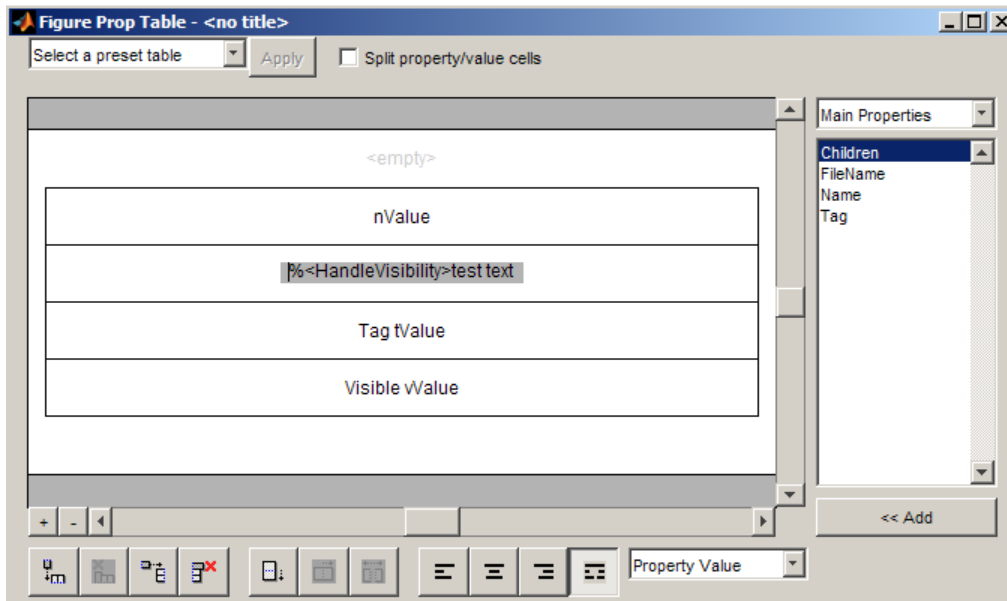
**Note** Table titles are always in nonsplit mode.

---

## Enter Text into Table Cells

For the text to be visible, the table must be in nonsplit mode. Clear **Split property/value cells**.

To enter text into the **HandleVisibility** table cell, double-click the cell. A gray box appears with the label for the cell property.




If you type text outside the angle brackets, the text appears as is in the report. Text inside the angle brackets must specify a valid property name. If you enter an invalid property name, the property name appears in the report without a property value.

## Add, Replace, and Delete Properties in Tables

### Adding Table Properties

To add a Handle Graphics property to a table, use the following steps.

- 1 In the Figure Property Table window, select a table row above which you want add a new property.
- 2 Click the Add Row Above Current Cell  button
  - a A new row appears above the current row.
- 3 Add the property to the new table row.
  - a Select the new table row.
  - b In the Properties Type drop-down list at the upper-right of the dialog box, select a property type.
  - c In the **Properties** list, select the property you want to add.
  - d Click the **<< Add** button, or double-click the property name. The property appears in the table row.

Alternatively, if you know the name of the property you want to add, enter the property name directly into the cell as described in “Enter Text into Table Cells” on page 6-10. For information about adding new table rows, see “Add and Delete Columns and Rows” on page 6-12.

## Replace Table Properties

To replace a property in a cell of a table in split mode, follow the instructions in “Adding Table Properties” on page 6-11.

---

**Note** You cannot use these steps to delete a property in a cell when the table is in nonsplit mode.

---

## Delete Table Properties

Delete a property by backspacing over it or using the **Delete** key.

## Format Table Columns, Rows, and Cells





### Add and Delete Columns and Rows

To add or delete a column or row, select a cell and then click one of the buttons described in the following table.

---

**Note** You cannot delete a row or column when it is the only row or column in the table.

---



Button	Action
	Add column (added to the left of the selected column)
	Delete selected column
	Add row (added above the selected row)
	Delete selected row


### Resize Columns

To resize the width of a column, click and drag its vertical borders as needed.

### Merge and Split Cells

To merge or split table cells, select a row and then click one of the buttons described in the following table.

Button	Action
	Merge cells downward
	Merge cells to the right

Button	Action
	Split cells



### Display or Hide Cell Borders

To toggle cell borders on and off:

- 1 Place your cursor in a cell and right-click to invoke its context menu.
- 2 Choose **Cell borders > Top, Bottom, Right, or Left** to toggle the specified border on or off.

### Zoom and Scroll

You can zoom in and out of the table with the zoom buttons, which are located to the left of the horizontal scroll bar.

Button	Action
	Zoom in
	Zoom out

You can scroll vertically and horizontally using the table scroll bars.

### Select a Table

To display property name/property value pairs, you can select a preset table or use a custom table.

- A preset table is built-in and formatted. You can select a preset table in the preset table selection list in the upper-left of the Figure Prop Table window. To apply a preset table, select the table and click **Apply**.
- To create a custom table, select a preset table and modify it to fit your needs by adding and/or deleting rows and properties. You may want to start with the **Blank 4x4** preset table.

---

**Note** You cannot save a custom table as a preset table. If you do so, you lose all changes to the custom table.

---

## Summary Table Components

### In this section...

“About Summary Table Components” on page 6-14

“Open the Example Report Template” on page 6-15

“Select Object Types” on page 6-15

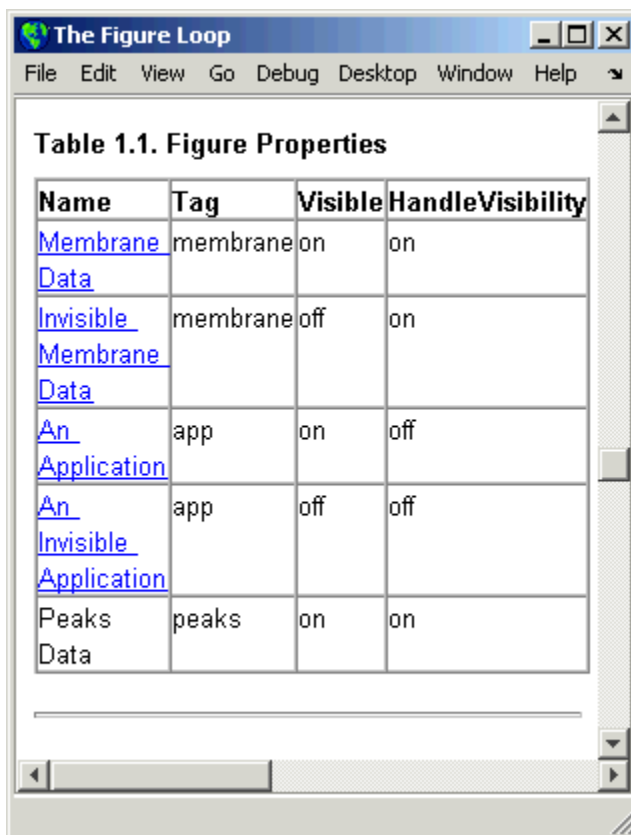
“Add and Remove Properties” on page 6-15

“Set Relative Column Widths” on page 6-16

“Set Object Row Options” on page 6-16

### About Summary Table Components

Summary table components insert tables that include specified properties for objects into generated reports. Summary tables contain one object per row, with each object property appearing in a column, as shown in the following summary table in the `figloop-tutorial` report.



The screenshot shows a window titled "The Figure Loop" with a menu bar (File, Edit, View, Go, Debug, Desktop, Window, Help). The main content area displays a table with the following data:

Name	Tag	Visible	HandleVisibility
<a href="#">Membrane Data</a>	membrane	on	on
<a href="#">Invisible Membrane Data</a>	membrane	off	on
<a href="#">An Application</a>	app	on	off
<a href="#">An Invisible Application</a>	app	off	off
Peaks Data	peaks	on	on

Many types of summary table components are available, including:

- Handle Graphics Summary Table
- Simulink Summary Table (requires Simulink Report Generator)
- Stateflow Summary Table (requires Simulink Report Generator)



The component used in this example represents MATLAB Report Generator summary table components, all of which exhibit similar behavior

## Open the Example Report Template

This example uses the `figloop-tutorial` report template. To open the figure loop tutorial report template, enter the following at the MATLAB command line:

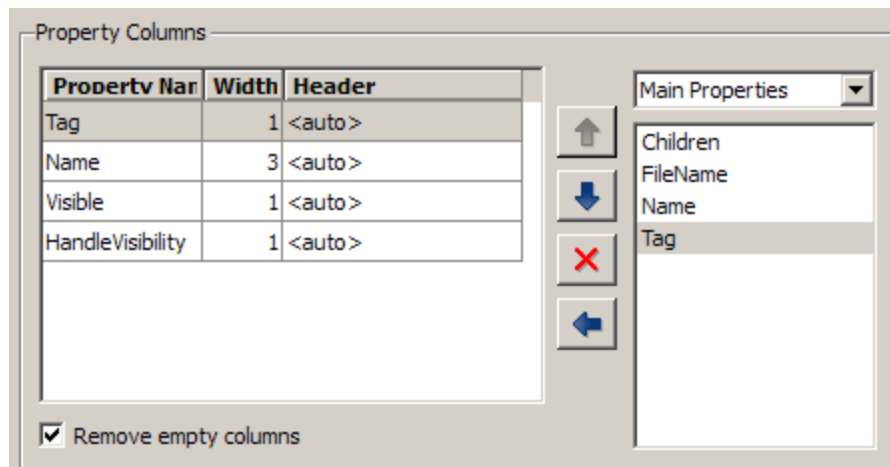
```
setedit figloop-tutorial
```

## Select Object Types


You can use the **Object type** selection list to choose Handle Graphics object types for the summary table, including blocks, signals, systems, and models. The `figloop-tutorial` reports on figure objects.

## Add and Remove Properties

You can select object properties to appear in the summary table from the Property Columns pane. To add a property to the summary table, select the property category from the property category drop-down box to the right of the **Property columns** table. Each property category has its own list of properties, which appear in the field under the box. The following figure shows **Main Properties** as the selected category.




To add a property:

- 1 Select the category from the property category drop-down box.
- 2 Select a property in the properties list.
- 3 Click the Add property  button.

The property appears in the **Property columns** table.

To remove a property from the table:

1 Select the property in the **Property columns** table.

2 Click the Delete property  button.

The property name is removed from the **Property columns** table.

---

**Note** After making changes in the Report Explorer, click **Apply** to make the changes take effect.

---

You can define your own properties by entering their names into the **Property columns** table using valid variable notation.

### Set Relative Column Widths

To apply a relative column width to the summary table columns in the generated report, double-click on the **Width** column of a row in the **Property columns** table . If you do not specify a value for this field, column widths automatically set.

### Set Object Row Options

You can use the Object Rows pane to set options for table rows, including anchor, filtering, and sorting options. Select **Insert anchor for each row** to place an anchor in each table row in the report. Use the **Include figures** list to specify what objects to include in the summary table.

Summary table components in `figloop-tutorial` report on figure objects. For more information on options for these figure objects, see the following sections:

- “Loop on the Current Figure” on page 6-19
- “Loop on Visible Figures” on page 6-20
- “Loop on Figures with Tags” on page 6-20

## Logical and Looping Components

Logical and looping components execute conditionally, determining when a child component executes or how many times a child component executes.

A looping component runs its child components a specified number of times. There are several looping components, such as logical loops and Handle Graphics loops.

For an example that uses loop components, see “Edit Figure Loop Components” on page 6-18.

## Edit Figure Loop Components

### In this section...

- “Figure Loop in a Report” on page 6-18
- “Figure Properties” on page 6-19
- “Loop on the Current Figure” on page 6-19
- “Loop on Visible Figures” on page 6-20
- “Loop on Figures with Tags” on page 6-20
- “Modify Loop Section Options” on page 6-20

### Figure Loop in a Report

This example uses the Figure Loop, which is representative of many types of loops. The Figure Loop component runs its child components several times. In each iteration, the Figure Loop applies its child components to Handle Graphics figures. The `figloop-tutorial` report setup file creates a report that documents several Handle Graphics figures.

- 1 At the MATLAB command prompt, enter:

```
setedit figloop-tutorial
```

- 2 To display the Handle Graphics figures, enter:

```
figloopfigures
```

The figures `Membrane Data`, `An Application`, and `Peaks Data` appear on the screen because their `visible` property is `'on'`. The `Invisible Membrane Data` and `An Invisible Application` figures do not appear on screen because their `visible` property is `'off'`. These invisible figures exist, but they are hidden.

- 3 In the Report Explorer, in the Outline pane on the left, select the Figure Loop component called **Figure Loop Section 1**.

The Properties pane for the Figure Loop component appears.

**FigureLoop**

Figure Selection

Include figures: All figures with tags: ▼

app  
membrane

Match with regular expressions

Loop Figure List

Membrane Data  
Invisible Membrane Data  
An Application  
An Invisible Application

Section Options

Create section for each object in loop

Display the object type in the section title

Create link anchor for each object in loop

## Figure Properties

Figure properties control which figures appear in the report. Table 1.1 of the `figloop-tutorial` report includes a summary of the properties of the figures used in this tutorial.

**Table 1.1. Figure Properties**

Name	Tag	Visible	HandleVisibility
<a href="#">Membrane Data</a>	membrane	on	on
<a href="#">Invisible Membrane Data</a>	membrane	off	on
<a href="#">An Application</a>	app	on	off
<a href="#">An Invisible Application</a>	app	off	off
Peaks Data	peaks	on	on

For this example, do not change these properties. For more information, see “Add, Replace, and Delete Properties in Tables” on page 6-11.

## Loop on the Current Figure

To include only the current figure in the report, from the **Include figures** list, select **Current figure only**. The current figure is the figure that is current when the report generates. This figure may not be the same figure that you selected as the current figure in the Report Explorer before report generation. For example, if the report generation process creates figures in your report, the last figure created with `HandleVisibility` set to 'on' is the current figure.

## Loop on Visible Figures

To include snapshots of all visible figures in your report, in the **Include figures** selection list, select **Visible figures**. This option inserts a snapshot and Property Table for all figures that are currently open and visible.

- 1 Select the **Data figures only (Exclude applications)** option to exclude figures from the loop whose `HandleVisibility` parameter is 'off'.
- 2 To generate the report, in the Report Explorer toolbar click the **Report** button.

In the generated report, scroll down to “Chapter 2 Figures in Report.” The Membrane Data and Peaks Data figures appear in the generated report.

## Loop on Figures with Tags

To include figures with specified tags in the report:

- 1 From the **Include figures** list, select **All figures with tags**.
- 2 In the list of tags, delete membrane.
- 3 Click **Report** to generate the report.

The An Application and An Invisible Application figures appear in the report. They both have an app tag.

## Modify Loop Section Options

In a loop, a *section* refers to a space in the generated report in which information, including text, images, and tables, appears. You can alter the appearance of sections in each loop appear in the report by using the options in the Figure Loop component's Section Options pane.

- **Create Section for Each Object in Loop** — Create an individual section for each object found in the loop, using the object title as the section title. This option is useful when a loop does not contain a **Chapter/Subsection** component that organizes the loop results.
- **Display the Object Type in the Section Title** — Precede section titles with object titles. Enable this option by selecting **Create section for each object in loop**. For example:
  - 1 Enter membrane back in the list of tags.
  - 2 Generate the figloop-tutorial report.

The figures produced by the loop are:

```
Membrane Data
Invisible Membrane Data
An Application
An Invisible Application
```

- 3 Enable the **Create section for each object in loop** option.
- 4 Enable the **Display the Object Type in the Section Title** option.
- 5 Generate the figloop-tutorial report.

The figures produced are now:

Figure - Membrane Data  
Figure - Invisible Membrane Data  
Figure - An Application  
Figure - An Invisible Application

The figures produced are now:

Figure - Membrane Data  
Figure - Invisible Membrane Data  
Figure - An Application  
Figure - An Invisible Application

- **Create a Link Anchor for Each Object in Loop** — Create a hyperlink to the object in the generated report.





# Template-Based Report Formatting

---

- “Report Templates” on page 7-2
- “Create a Report Template” on page 7-6
- “Copy a Template” on page 7-7
- “Set a Template’s Properties” on page 7-8
- “Open a Template” on page 7-9
- “Generate a Report Using a Template” on page 7-10
- “Default Template Contents” on page 7-11
- “Customize Microsoft Word Report Styles” on page 7-16
- “Customize Microsoft Word Component Templates” on page 7-18
- “Customize a Microsoft Word Title Page Template” on page 7-25
- “Create a Custom HTML or PDF Template” on page 7-29

## Report Templates

### In this section...

“Template-Based Output Types” on page 7-2

“Templates Versus XSL and DSSSL Style Sheets” on page 7-3

“Component Styles” on page 7-3

“Component Templates” on page 7-4

“Component Holes” on page 7-4

“Template Cache” on page 7-5

The Report Explorer allows you to use templates to format your reports. A template is an HTML or Word document that specifies the fixed content, format, and layout of your report. You can use Microsoft Word templates to format Word and PDF reports and HTML templates to format HTML and PDF reports. The MATLAB Report Generator comes with a default set of Word templates for PDF and Word reports and HTML templates for PDF and HTML reports. You can create your own templates by copying and customizing these default templates.

### Template-Based Output Types

The Report Explorer provides a set of template-based report output types. These output types appear in the **File format** drop-down list in your report generator’s root **Form** or **Reporter** component. Selecting a template-based output type populates the adjacent drop-down list with the templates available for that output type, including the Report Explorer’s built-in templates and any templates that existed on the MATLAB path or in the current directory at the start of your MATLAB session. By default the Report Explorer uses a default template to format a report for the specified output type. You can specify another template by selecting from the adjacent list.

## Template-Based Output Types

Name on Form Component	Name on Report Component	Generates
Direct PDF	Direct PDF (from template)	Formatting Objects (FO) representation of a report's content and format based on an HTML template with PDF-specific extensions. The FO representation is converted to PDF.
PDF (from Word)	PDF (from Word template)	Microsoft Word report from a Microsoft Word template and then uses Microsoft Word to convert the Word report to PDF. This output type is available only on Windows. Report size is limited by Word's pagination capacity, typically about 500 pages.
HTML	HTML (from template)	HTML report package from an HTML template package. Both the report and the template are zip files that contain HTML documents, style sheets, images, and JavaScripts.
Single-File HTML	Single-File HTML (from template)	HTML report from an HTML template package. The HTML report is a single HTML file that embeds the report's text, style sheets, images, and JavaScripts.
Word	Word (from template)	Microsoft Word report from a Word report.

## Templates Versus XSL and DSSSL Style Sheets

Besides templates, the Report Explorer supports use of XSL or DSSSL style sheets to format reports. An XSL style sheet is a program, written in a dialect of XML called XSL, that converts and formats your report's intermediate XML content to HTML or PDF output. A DSSSL style sheet is a program, written in a dialect of Lisp, that converts your report's intermediate XML content to Microsoft Word output. The Report Explorer supports style-sheet-based formatting to provide backward compatibility with report generators developed with the Report Explorer in releases that preceded availability of templates. You should use templates exclusively for new report generators that you develop with the Report Explorer. This is because templates have significant advantages over style sheets:

- **Formatting:** XSL and DSSSL style sheets have limited formatting options. For example, all titles and all body paragraphs in your report must have the same format. By contrast, templates allow you to use all formatting options available in Microsoft Word and HTML documents.
- **Scalability:** The Report Explorer uses a MATLAB-based file converter, called `db2dom`, to convert reports based on templates. The Report Explorer uses Java-based file converters, `xslt` and `JADE`, to convert reports based on XSL and DSSSL style sheets. The `db2dom` converter is typically an order of magnitude faster than `xslt` and `JADE` and uses no Java memory for Word and HTML output and much less memory for PDF output. As a result, the Report Explorer can generate much larger reports with templates than with style sheet-based output.

## Component Styles

Every report template contains a style sheet. This style sheet, not to be confused with an XSL or DSSSL style sheet, is a document that defines named sets of text, paragraph, list, and table formats called styles. During report generation, the Report Explorer's file converter, `db2dom`, copies the template style sheet into the generated report and assigns the style names to paragraphs, text, lists,

and tables generated by your report generator's components. The program that you use to display or print your report, such as an HTML browser or Microsoft Word, uses the styles to format your report.

The Report Explorer's default templates defines all the styles needed to format a report generated by the Report Explorer from your report setup file. To distinguish them from other styles, the names of these styles begin with the prefix, **rg**, for example, **rgParagraph**. You can modify the appearance of a report by customizing the definitions (but not the names) of these styles in a copy of a default template and using the copy to generate the report.

You can also define your own styles in a customized template and assign them to components whose dialogs contains a **StyleName** property. Components that have a **StyleName** property include the **Text** and **Paragraph** components. In this way, you can customize the appearance of individual instances of a component. For example, the default style of a **Paragraph** component is **rgParagraph**. By creating and assigning your own style, for example, **myParagraph**, to a particular **Paragraph** component, you can differentiate the appearance of this component's output from that of paragraphs that have the default **rgParagraph** style.

## Component Templates

The Report Explorer's default templates contain a component template library. A component template library is a document that defines templates for Report Explorer components, such as the **Title Page** component and the **Chapter/Section** component. Each template in the library has a name, for example, **rgRectoTitlePage**, which enables the Report Generator to locate the component template in the library. The component library allows a single template, called the main template, to contain all the templates needed to format a report generated from the main template.

You can change the format of a report component, such as a title page, by customizing its template in a copy of a default template and using that customized template to generate the report. You can also create and store templates for **Subform** components in a **Form** component's main template. In this way, you can create custom report components with custom content and custom formats. For example, you can create a custom title page template and use it with a **Subform** component to generate a title page that contains content not defined by the **Title Page** component, such as a sign-off block.

## Component Holes

The Report Explorer's default component templates contain placeholders, called holes, that designate where to insert generated content relative to the template's fixed content and other generated content. For example, the **Title Page** component's templates contain holes for a report's title, subtitle, author, abstract, etc. The Report Explorer replaces these holes with generated content during report generation. For example, it replaces the title hole in the **Title Page** template with the title specified by the **Title Page** component's Title property.

You can alter the layout and content of a component that specifies content by rearranging or deleting its holes. For example, you can delete or adjust the location of a title page's subtitle by moving or deleting the subtitle hole in the **Title Page** component's template. You can also include holes in the templates that you create for **Form** and **Subform** components and fill those holes using **Template Hole** components in your report generator setup file. In this way, for example, you can generate a title page that exactly fits your title page layout and format requirements.

## Template Cache

The first time you open the **Report Explorer** in a MATLAB session it searches the MATLAB path for templates. It stores all the templates that it finds in a cache. It also adds any templates that you create in the session to the cache. Subsequently it searches the cache for any template that you specify that is not in the current directory. This avoids the need to search the MATLAB path every time the Report Explorer needs to generate a report based on a template. If you try to use a template that is not on the MATLAB path at the beginning of the MATLAB session, the **Report Explorer** indicates that it cannot find the template. In this case, you can either change MATLAB working directory to the directory to the template directory or you can add the directory to the MATLAB path and refresh the cache. To refresh the cache, execute

```
>> rptgen.db2dom.TemplateCache.getTheCache(true);
```

at the MATLAB command line.

## See Also

### Related Examples

- “Create a Report Template” on page 7-6
- “Generate a Report Using a Template” on page 7-10
- “Customize Microsoft Word Report Styles” on page 7-16
- “Customize Microsoft Word Component Templates” on page 7-18
- “Customize a Microsoft Word Title Page Template” on page 7-25
- “Create a Custom HTML or PDF Template” on page 7-29

### More About

- “Default Template Contents” on page 7-11

## Create a Report Template

To create a report template for use with the Report Explorer:

- 1** Copy an existing template, either one of the Report Explorer's default templates or a template based on one of the default templates.
- 2** Set the template's properties.
- 3** Open the template in a template editor.
- 4** Edit the template to meet your requirements.
- 5** Save the template.

### See Also

#### Related Examples

- "Copy a Template" on page 7-7
- "Set a Template's Properties" on page 7-8
- "Open a Template" on page 7-9
- "Customize Microsoft Word Report Styles" on page 7-16
- "Customize Microsoft Word Component Templates" on page 7-18
- "Customize a Microsoft Word Title Page Template" on page 7-25

#### More About

- "Report Templates" on page 7-2

## Copy a Template

- 1 In Report Explorer, select **Tools > Edit Document Conversion Template**.
- 2 In the template library (center) pane, select the template you want to copy. For example, select the **Default HTML Template**.
- 3 In the template dialog (right) pane, click **Copy template**. A file browser appears.
- 4 In the file browser, navigate to a folder where you want to save the template. The folder should be on the MATLAB path (for example, in the MATLAB folder in your home folder).
- 5 Specify the file name, using the default file extension for the kind of template you are copying: a Microsoft Word template (.dotx), an HTML template (.htmlx), single-file HTML template (.html), or PDF template (.pdfx).
- 6 Click **Save**. The new template appears in the list of templates in the Report Explorer's center pane.
- 7 Select the new template. The template's dialog appears in the right pane. The initial name of the copy in the list of templates is **Copy of ORIGINAL**, where **ORIGINAL** is the name of the template that you copied.
- 8 Set the new template's properties.

## Set a Template's Properties

### In this section...

"Set Template Properties Interactively" on page 7-8

"Set Template Properties Programmatically" on page 7-8

A conversion template's properties allows the Report Explorer to keep track of the templates that you create. You can set the properties interactively or programmatically.

### Set Template Properties Interactively

- 1 In Report Explorer, select **Tools > Edit Document Conversion Template** to display the Report Explorer's template library (if it is not already displayed).
- 2 From the list of templates in the Report Explorer's center pane, select the template whose properties you want to set. The template's dialog appears in the right pane displaying the template's current properties.
- 3 In the dialog box, enter values for the template's properties in the following fields:

<b>Template id</b>	Unique value used by the Report Explorer to identify this template.
<b>Display name</b>	Appears in the drop-down list of templates in the <b>Form</b> and <b>Report</b> component dialogs.
<b>Description</b>	Describes this template's purpose.
<b>Creator</b>	Specifies this template's creator.

- 4 To save the template properties you entered, click outside of the Report Explorer's dialog pane.

### Set Template Properties Programmatically

Use the DOM API's `mlreportgen.dom.Document.getCoreProperties` and `mlreportgen.dom.Document.setCoreProperties` methods to get and set a conversion template's properties. Set the template's core properties as follows to correspond to the properties that appear on the template's dialog in the Report Explorer:

Dialog Property	Core Property
<b>Template id</b>	Identifier
<b>Display name</b>	Title
<b>Description</b>	Description
<b>Creator</b>	Creator



## Open a Template

You can open a custom template to edit it. Opening a Word template opens the template in Microsoft Word. Opening a single-file HTML template opens the file in the HTML editor specified by your Report Generator preferences (the MATLAB editor by default). Opening an HTML or PDF packaged template unzips the template and then opens the main template document and the document part template library in the HTML editor. To learn more about contents of templates, see “Default Template Contents” on page 7-11.

You cannot edit a default template. If you want to customize a default template, you must create a copy of the template and edit the copy. See “Copy a Template” on page 7-7.

---

**Tip** The Report Generator repackages an open HTML or PDF template before running a report based on the template. Run a report after editing an HTML or PDF template to ensure that your changes are saved.

---

- 1 In Report Explorer, select **Tools > Edit Document Conversion Template**.
- 2 Select a custom template from the list of templates in the Report Explorer’s template list (center) pane. The template’s dialog appears in the right pane.

---

**Note** If your custom template does not appear in the list, refresh the Report Explorer’s template cache (see “Template Cache” on page 7-5).

---

- 3 Click **Open template** to open the report’s template and component template library for PDF and HTML templates.
- 4 Click **Open style sheet** to open the template’s style sheet.

### See Also

#### Related Examples

- “Edit HTML or PDF Styles in a Template” on page 7-30
- “Create a Report Template” on page 7-6


#### More About

- “Report Templates” on page 7-2

## Generate a Report Using a Template

- 1 In Report Explorer, in the outline pane, select the top-level component of the report.
- 2 In the dialog box, set **File format** to one of these options, depending on whether the root component is a **Form** component or a **Report** component:

Form Component	Report Component
Direct PDF	Direct PDF (from template)
HTML	HTML (from template)
PDF (from Word)	PDF (from Word template)
Single-File HTML	Single-File HTML (from template)
Word	Word (from template)

- 3 Optionally, from the list of templates available for the current file format, select a template.
- 4 If you select HTML or HTML (from template) for the file format, choose a packaging option for the output files.
  - **Unzipped** — Generate the report files in a subfolder of the current folder. The subfolder has the report name.
  - **Zipped** — Package report files in a single compressed file that has the report name, with a .zip extension.
  - **Both Zipped and Unzipped**
- 5 In the toolbar, click the **Report** button .

### See Also

### More About

- “Report Templates” on page 7-2

## Default Template Contents

<b>In this section...</b>
“Default Styles” on page 7-11
“Component Templates” on page 7-13

A default report template contains an empty main document that is copied to the report output to hold the report content during report generation. In addition, it contains:

- A style sheet with default style definitions for report elements.
- A template library that contains templates for the title page, table of contents, chapters, and chapter and section titles. The templates define page headers and footers, page size, page margins, and page orientation for a report’s title page, table of contents, and chapters. The templates also include toc and autonumber fields used by Microsoft Word and HTML browsers to generate table of contents and chapter and section numbers when they open a report.

### Default Styles

A default report template includes styles that the Report Explorer uses to format components during report generation. Most styles begin with `rg` (for example, `rgTitle`). Styles for syntax highlighting MATLAB code begin with `MWSG`, for example, `MWSHKeywords`. The default style names and formatting are the same for the Word and HTML templates, to the extent applicable. For example, page break formatting applies when you use a Word template or an HTML template used for formatting PDF reports, but not an HTML template used for formatting HTML reports..

You can modify the built-in styles, but do not delete them. In addition, you can define your own styles and use them in components that allow you to specify a style, such as the **Paragraph** component.

Style	Component to which the style applies
<code>MWSHComment</code>	MATLAB code comment
<code>MWSHKeywords</code>	MATLAB code keywords
<code>MWSHstrings</code>	MATLAB code character vectors
<code>rgAbstract</code>	<b>Title Page</b> component abstract
<code>rgAbstractTitle</code>	<b>Title Page</b> component abstract section
<code>rgAuthor</code>	<b>Title Page</b> component front page author
<code>rgAuthorVerso</code>	<b>Title Page</b> component back page author
<code>rgBody</code>	<b>Text</b> component
<code>rgChapter</code>	<b>Chapter</b> component
<code>rgChapterTitle</code>	<b>Chapter</b> component title
<code>rgCopyright</code>	<b>Title Page</b> component copyright
<code>rgFigure</code>	<b>Paragraph</b> that contains an image generated by a snapshot or <b>Image</b> component, to adjust the spacing of images relative to adjacent paragraphs
<code>rgFigureCaption</code>	The caption for <b>Image</b> component and snapshot components
<code>rgFigureTitle</code>	The <b>Image</b> component and snapshot components title

Style	Component to which the style applies
rgFigureTitleNumber	The <b>Image</b> component and snapshot components title number
rgFigureTitlePrefix	The <b>Image</b> component and snapshot components title prefix
rgFigureTitleText	The <b>Image</b> component and snapshot components title text
rgLegalNotice	<b>Title Page</b> component legal notice section
rgListStyle	Specifies the style of lists generated by the <b>List</b> component.
rgListTitle	The <b>List</b> component title
rgListTitleNumber	The <b>List</b> component title number
rgListTitlePrefix	The <b>List</b> component title prefix
rgListTitleText	The <b>List</b> component title text
rgParagraph	<b>Paragraph</b> component text
rgParagraphTitle	<b>Paragraph</b> component title
rgProgramListing	Code generated by: <ul style="list-style-type: none"> <li>• <b>Text</b> component with <b>Show text as syntax highlighted MATLAB code</b> option is selected</li> <li>• <b>MATLAB Function Block</b> component</li> <li>• <b>Truth Table</b> component</li> </ul>
rgPubDate	<b>Title page</b> report creation date
rgPubDatePrefix	<b>Title page</b> report creation date prefix
rgSect1Title	<b>Section</b> title for first-level section in a chapter
rgSect1TitleNumber	Number for <b>Section</b> title for first-level section in a chapter
rgSect1TitlePrefix	Prefix for <b>Section</b> title for first-level section in a chapter
rgSect1TitleText	Text for <b>Section</b> title for first-level section in a chapter
rgSect2Title	<b>Section</b> title for second-level section in a chapter
rgSect2TitleNumber	Number for <b>Section</b> title for second-level section in a chapter
rgSect2TitlePrefix	Prefix for <b>Section</b> title for second-level section in a chapter
rgSect2TitleText	Text for <b>Section</b> title for second-level section in a chapter
rgSect3Title	<b>Section</b> title for third-level section in a chapter
rgSect3TitleNumber	Number for <b>Section</b> title for third-level section in a chapter
rgSect3TitlePrefix	Prefix for <b>Section</b> title for third-level section in a chapter
rgSect3TitleText	Text for <b>Section</b> title for third-level section in a chapter
rgSect4Title	<b>Section</b> title for fourth-level section in a chapter
rgSect4TitleNumber	Number for <b>Section</b> title for fourth-level section in a chapter
rgSect4TitlePrefix	Prefix for <b>Section</b> title for fourth-level section in a chapter
rgSect4TitleText	Text for <b>Section</b> title for fourth-level section in a chapter
rgSect5Title	<b>Section</b> title for fifth-level section in a chapter
rgSect5TitleNumber	Number for <b>Section</b> title for fifth-level section in a chapter
rgSect5TitlePrefix	Prefix for <b>Section</b> title for fifth-level section in a chapter

<b>Style</b>	<b>Component to which the style applies</b>
rgSect5TitleText	Text for <b>Section</b> title for fifth-level section in a chapter
rgSubTitle	<b>Title Page</b> component subtitle
rgTable	<b>Table</b> content
rgTableTitle	<b>Table</b> title
rgTableTitleNumber	<b>Table</b> title number
rgTableTitlePrefix	<b>Table</b> title prefix
rgTableTitleText	<b>Table</b> title text
rgTitle	<b>Title Page</b> component front page title abstract, and legal notice section
rgTitleVerso	<b>Title Page</b> component back page title abstract, and legal notice section
rgTOCSection	Table of contents

## Component Templates

The component template library of a default report template contain templates to determine the page layout and formats of report sections, such as the title page, table of contents, and chapters.

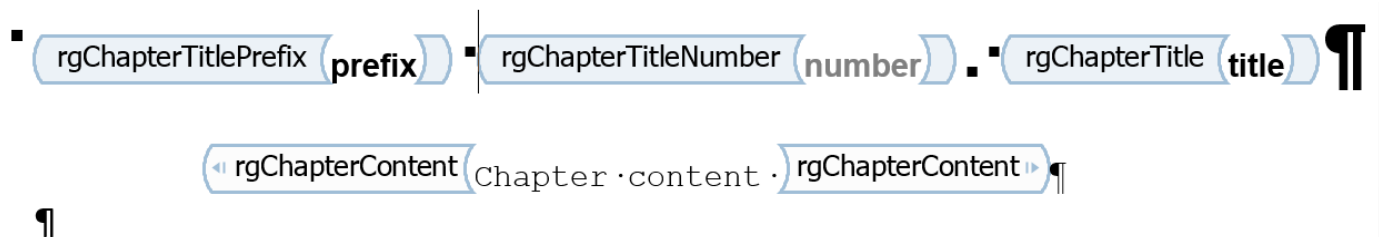
<b>Part Template</b>	<b>Report Explorer Components the Part Template Formats</b>
rgRectoTitlePage	<b>Title Page</b> Front title page contents, including the report title, subtitle, author, and an image.
rgVersoTitlePage	<b>Title Page</b> Back title page contents, including the date published, copyright, legal notice, and abstract.
rgTOCSectionTitle	The table of contents automatically generated for Word and PDF.
rgChapter	<b>Chapter/Section</b> Chapter (top-level section), including the title (prefix, such as <b>Chapter</b> , number, and title) and for the content.
rgSect1Title	<b>Chapter/Section</b> Title for a section (sections below the chapter level). The title can include a prefix (such as <b>Chapter</b> ), number, and title.
rgSect2Title	
rgSect3Title	
rgSect4Title	
rgSect5Title	

Part Template	Report Explorer Components the Part Template Formats
rgListTitle	<b>List</b> Title of the list.
rgTableTitle	<b>Table, Array-BasedTable</b> , and table components such as <b>Handle Graphics Property Table</b> Table title, including prefix (such as <b>Table</b> , number, and title) and for the content.
rgFigureTitle	<b>Table and Array-BasedTable</b> Table title, including prefix (such as <b>Table</b> , number, and title) and for the content.

### Component Template Holes

Component templates include fill-in-the-blanks hole markup. The Report Explorer fills holes with content that components generate.

For example, the `rgChapter` template in the default Word template includes an `rgChapterTitle` hole.



The Report Explorer fills the `rgChapterTitle` hole with the contents of the `Title` property of top-level **Chapter/Section** components in a report.

You can rearrange or delete holes to change the order in which generated content appears in the report or to omit content. Do not add holes for content that a component does not specifically address. If you add holes, the Report Explorer ignores them. If you need to add content to a **Title Page** or other component, use a **Form** or **Subform** component instead.

### See Also

#### Related Examples

- “Generate a Report Using a Template” on page 7-10
- “Copy a Template” on page 7-7
- “Customize Microsoft Word Report Styles” on page 7-16
- “Customize Microsoft Word Component Templates” on page 7-18
- “Customize a Microsoft Word Title Page Template” on page 7-25

## **More About**

- “Report Templates” on page 7-2

## Customize Microsoft Word Report Styles

<b>In this section...</b>
---------------------------

“Customize Default Microsoft Word Component Styles” on page 7-16
--

“Create Styles in a Microsoft Word Template” on page 7-16
---

You can customize report styles in a custom Word template or add styles to a custom Word template.

For more information about Word styles, see the Microsoft Word documentation.

### Customize Default Microsoft Word Component Styles

---

**Note** You cannot customize a default template’s styles directly. You must create a copy of the default template and customize the copy’s styles, see “Copy a Template” on page 7-7.

---

- 1 In the Report Explorer, select **Tools > Edit Document Conversion Template**.
- 2 From the list of templates, select the custom template that contains the style you want to customize.

---

**Note** If the template does not appear in the template list, refresh the Report Explorer’s template cache, see “Template Cache” on page 7-5.

---

- 3 In the dialog box, click **Open style sheet**. The Microsoft Word **Manage Styles** dialog box appears.
- 4 Use the **Manage Styles** dialog box to modify or create styles.

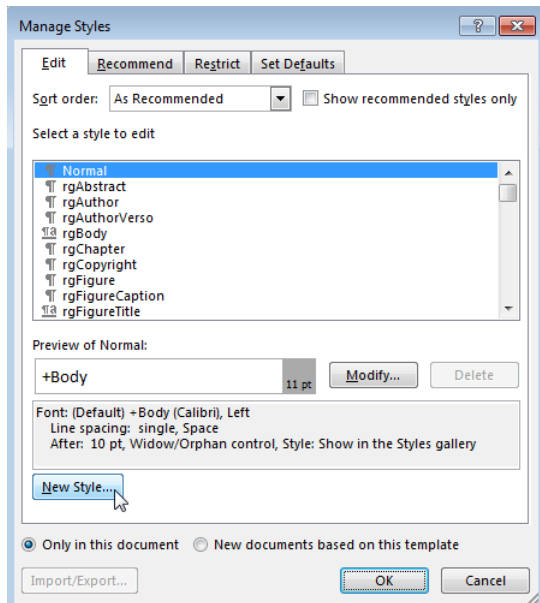
Styles that begin with rg (for example, rgParagraph) are the default styles used for report components. A default style applies to all instances of a component with which it is associated. (In the Report Explorer, some components allow you to replace the name of a default style with the name a style that you create. You can then specify different styles for different instances of the same component.)

- 5 Close the **Manage Styles** dialog box.
- 6 Save the template.

### Create Styles in a Microsoft Word Template

- 1 In the Report Explorer, select **Tools > Edit Document Conversion Template**.
- 2 From the list of templates, select a custom template.
- 3 In the dialog box, click **Open style sheet**.
- 4 If applicable, select an existing style to use as a starting point for the new style.
- 5 Click **New Style** .





- 6 Specify a name for the new style and define the style characteristics. To save the new style definition, click **OK** and close the dialog box.
- 7 In the Manage Styles dialog box, click **OK**.
- 8 In Word, save and close the template.

## See Also

### Related Examples

- “Customize Microsoft Word Component Templates” on page 7-18
- “Customize a Microsoft Word Title Page Template” on page 7-25

### More About

- “Report Templates” on page 7-2
- “Default Template Contents” on page 7-11

## Customize Microsoft Word Component Templates

In this section...
“Custom Word Component Templates” on page 7-18
“Display the Developer Ribbon in Word” on page 7-18
“Customize a Word Component Template” on page 7-19
“Set Default Text Style for a Hole” on page 7-19
“Distinguish Inline and Block Holes” on page 7-21
“Avoid Changing Block Holes to Inline Holes” on page 7-21
“Delete a Hole” on page 7-22
“Add an Inline Hole” on page 7-23
“Add a Block Hole” on page 7-23
“Remove or Modify Chapter Prefix” on page 7-23

### Custom Word Component Templates

You can customize a Word component template (such as a title page template) to:

- Tailor report formatting to meet your specific formatting requirements.
- Delete content by deleting a hole. For a description of holes, see “Component Holes” on page 7-4.
- Change component layout by changing the order of holes.
- Change a title page, table of contents, or chapter page layout, including page size, page orientation, page margins, and header and footer content and format.

If you delete a hole in a component template, the generated report does not include the component data associated with that hole. For example, the `rgRectoTitlePage` part template includes an `rgAuthor` hole. If you delete the `rgAuthor` hole, then reports generated with the template do not include the author, even if the report has a **Title Page** component that specifies a value for the `Author` property.

The Report Explorer ignores holes in a component template that do not correspond to content properties specified by the component. If you need to generate content not addressed by one of the Report Explorer’s components, use a **Form** or **Subform** component to generate the content. For example, you can use a Subform component with a custom template to generate a title page that contains content, such as a sign-off block, not addressed by the **Title Page** component.

### Display the Developer Ribbon in Word

To work with holes in a Word template, use the Word **Developer** ribbon. If the **Developer** tab is not showing in your Word ribbon, add it to the ribbon.

- 1 In Word, select **File > Options**.
- 2 In the Word Options dialog box, select **Customize Ribbon**.
- 3 In the **Customize the Ribbon** list, select the **Developer** check box and click **OK**.

---

**Tip** If you do not see **Developer** check box in the list, set **Customize the Ribbon** to Main Tabs.

---

## Customize a Word Component Template

To customize a report element such as a title page, replace the appropriate default component template with a customized copy of the template. For an example of creating a custom Word component template, see “Customize a Microsoft Word Title Page Template” on page 7-25.

---

**Note** You cannot customize a default template’s styles directly. You must create a copy of the default template and customize the copy’s styles, see “Copy a Template” on page 7-7.


---

- 1 In the Report Explorer, select **Tools > Edit Document Conversion Template**.
- 2 In the list of templates in the middle pane, select a custom template.

---


**Note** If the template does not appear in the template list, refresh the Report Explorer’s template cache, see “Template Cache” on page 7-5.

---

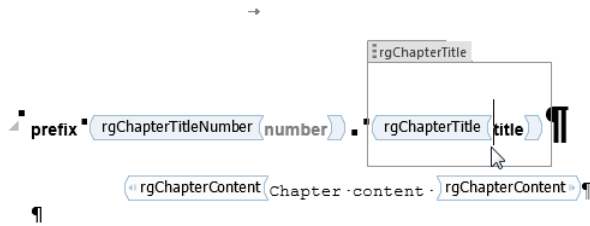
- 3 In the **Properties** pane, click **Open template**.
- 4 At the beginning of the template, position the cursor in the first paragraph and click the **Quick Parts** button.
- 5 In the **Insert** tab, select the **Quick Parts**  button.
- 6 In the Quick Parts Gallery, select the part template (for example, rgChapter).
- 7 Edit the copy of the part template. For example, remove a hole by right-clicking and selecting **Remove Content Control**.
- 8 In the template, select the part template, including all of its holes.
- 9 In the Quick Parts Gallery, select **Save Selection to Quick Part Gallery**.
- 10 In the Create New Building Block dialog box, set **Name** to the part template name (for example, rgChapter) and the **Category** to ml reportgen. Click **OK**.
- 11 In the template, delete the customized part template.
- 12 Save the main template.

## Set Default Text Style for a Hole

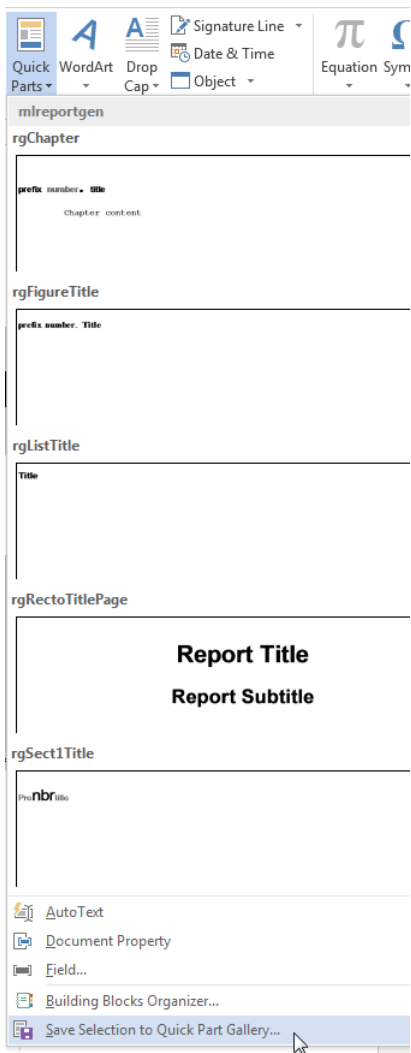
Your template can specify the name of a style to use as a default to format text generated for a hole.

- 1 In the Report Explorer, select **Tools > Edit Document Conversion Template**.
- 2 In the list of templates in the middle pane, select the custom template that has the hole you want to set the default text style for.
- 3 In the Template Browser, click **Open template**.
- 4 In the **Insert** tab, select the **Quick Parts**  button.
- 5 In the Quick Parts Gallery, select the part template that contains the hole (for example, rgChapter).

- 6 Right-click in the text area of the hole whose default text style you want to specify. For example, in rgChapter, right-click in the rgChapterTitle hole.



- 7 Select **Properties**.
- 8 In the Content Control Properties dialog box, select the **Use a style to format text typed into the empty control** check box.
- 9 From the **Style** list, select a style to use an existing style or select **New Style** to create a new style to use as the default style and click **OK**.
- 10 Select the part template and click the **Quick Parts** button.
- 11 Click **Save Selection to Quick Part Gallery**.



- 12 In the Create New Building Block dialog box, set **Name** to the part template name (for example, rgChapter) and the **Category** to m\reportgen. Click **OK**.
- 13 Save and close the template.

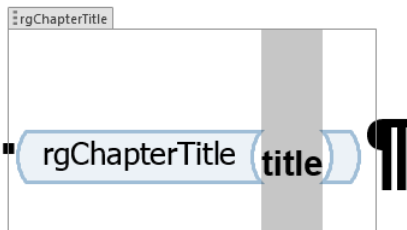
## Distinguish Inline and Block Holes

The Report Explorer supports two types of holes: inline and block.

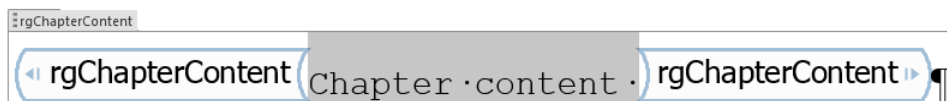
- Use an inline hole is for content that you can include in a Word paragraph.
- Use a block hole for content that you cannot embed in a paragraph.

You can configure the Word editor to provide visual cues that indicate whether a hole is an inline or block hole.

- 1 Open the custom Word template.
- 2 On the Word ribbon, select the **Home** tab.
- 3 Click the **Show/Hide** ¶ button to display Word paragraph markers.
- 4 On the Word ribbon, select the **Developer** tab.
- 5 Click **Design Mode** to see the Word markup for holes.
- 6 Click a hole to determine whether it is an inline or block hole.
  - Inline hole — The bounding box does not include the paragraph marker.



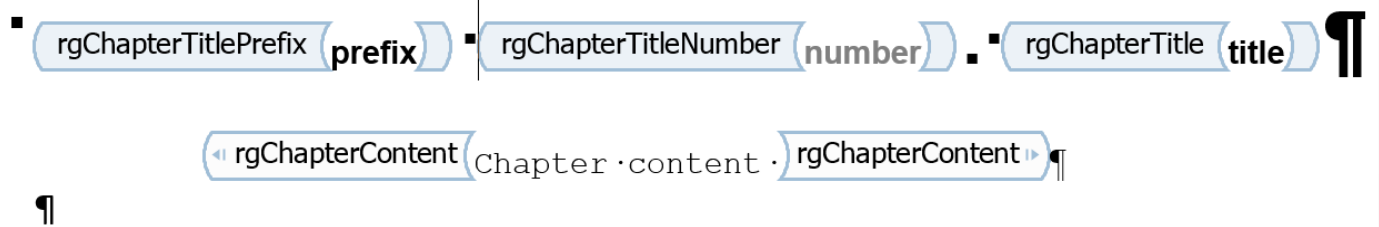
- Block hole — The bounding box does include the paragraph marker.



## Avoid Changing Block Holes to Inline Holes

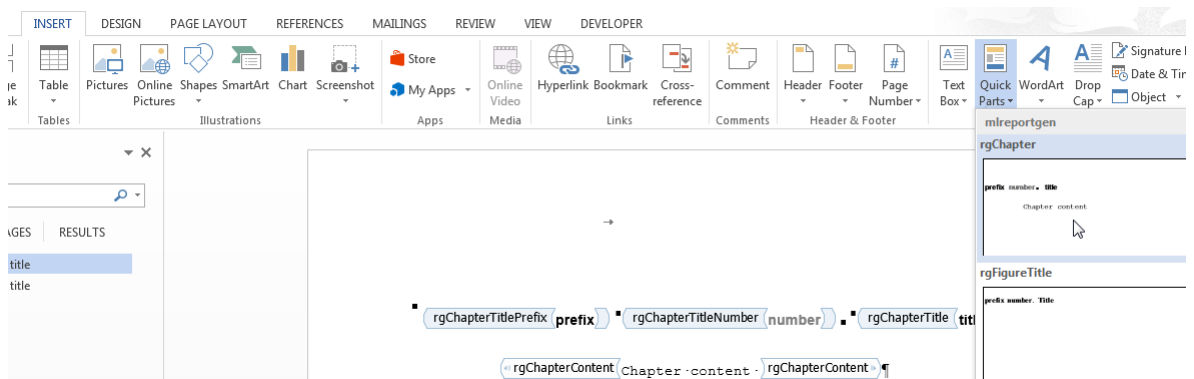
Do not change a block hole to an inline hole.

You can accidentally change a block hole to an inline hole by removing the paragraph marker of an inline hole that is followed by a block hole. For example, if you delete the paragraph marker for the rgChapterTitle inline hole, the rgChapterContent block hole changes to an inline hole.



## Delete a Hole

- 1 In the Report Explorer, select **Tools > Edit Document Conversion Template**.
- 2 In the list of Word templates in the middle pane, select the custom template that you want to edit.
- 3 In the Template Browser, click **Open template**.
- 4 To display Word paragraph markers (if they are not already visible), on the Word ribbon in the **Home** tab, click the **Show/Hide ¶** button.
- 5 In the Word ribbon, in the **Insert** tab, click the **Quick Parts ¶** button.
- 6 Select the part template to customize. For example, select rgChapter to customize the part template for a chapter.




**Tip** To display Word markup for the part template, on the Word ribbon, in the **Developer** tab, click **Design Mode**.

- 7 Write down the name of the part template you are customizing, because you need to enter that name later in this procedure.
- 8 In the rgChapter part template, delete the rgChapterTitlePrefix hole. Select the hole markup and click the **Delete** key.
- 9 In the template, select all of the contents of the part template.
- 10 Right-click and select **Properties**.
- 11 In the Content Control Properties dialog box, in the **Title** and **Tag** fields, enter the name of the template part you are customizing rgChapter. Click **OK**.
- 12 In the template, select all of the contents of the part template. In the **Insert** tab, click the **Quick Parts** button.
- 13 Click **Save Selection to Quick Part Gallery**.

- 14 In the Create New Building Block dialog box, set **Name** to the part template name (for example, rgChapter) and the **Category** to m\reportgen. Click **OK**.
- 15 In the template, select all of the contents of the part template and click the **Delete** button.
- 16 Save and close the template.


## Add an Inline Hole

- 1 In the Report Explorer, select **Tools > Edit Document Conversion Template**.
- 2 In the list of Word templates in the middle pane, select the custom template that you want to edit.
- 3 In the Template Browser, click **Open template**.
- 4 To display Word paragraph markers, click the **Show/Hide**  button.
- 5 Position the Word insertion mark at the point in a paragraph where you want to add an inline hole.

---

**Tip** If the hole is the only content in a paragraph or is at the end of a paragraph, add several blank spaces and insert the hole before the spaces.

---

- 6 Click the **Rich Text Control** button . Word inserts a rich text control at the insertion point.
- 7 To see hole markup, on the Word ribbon, in the **Developer** tab click **Design Mode**.
- 8 Right-click in the hole and select **Properties**.
- 9 In the dialog box, in the **Title** and **Tag** fields, enter the name of the hole. Use a Report Explorer hole name. For example, if you insert an rgChapterTitlePrefix hole, set the **Title** and **Tag** fields to rgChapterTitlePrefix.
- 10 In the template, select all of the contents of the part template. In the **Insert** tab, click the **Quick Parts** button.
- 11 Click **Save Selection to Quick Part Gallery**.
- 12 In the Create New Building Block dialog box, set **Name** to the part template name (for example, rgChapter) and the **Category** to m\reportgen. Click **OK**.
- 13 In the template, select all of the contents of the part template and click the **Delete** button.
- 14 Save and close the template.

## Add a Block Hole

Creating a block-level hole in a Word document is essentially the same as creating an inline hole. The main difference is that rich text content control must contain an (empty) paragraph instead of residing in a paragraph. Create an empty paragraph at the point where you want to create a block-level hole. If you are at the end of a document, create a second empty paragraph.

## Remove or Modify Chapter Prefix

Reports that use the default templates include the word **Chapter** as a prefix in the chapter title. If you do not want to use the prefix, you can delete from your template before you create a report. If you want to use a word other than **Chapter**, for example, for localization, you can replace the prefix.

- 1 Open the template.
- 2 On the Word **Insert** ribbon, in the **Text** area, click the **Explore Quick Parts** button.
- 3 To insert an instance of the part you want to modify in your template, select the **rgChapter** quick part.
- 4 Edit the instance. You can remove the prefix hole, or you can replace it with fixed text.

Make sure that the style applied to this line is still **rgChapter**.



- 5 Select the edited instance. Then, on the **Insert** ribbon, click the **Explore Quick Parts** button and select **Save Selection to Quick Parts Gallery**.
- 6 In the dialog box, set **Name** to rgChapter and **Category** to m\reportgen, and then click **OK**. Confirm that you want to overwrite the previous version.
- 7 Save and close the template.

## See Also

### Related Examples

- "Create a Report Template" on page 7-6
- "Customize Microsoft Word Report Styles" on page 7-16
- "Customize a Microsoft Word Title Page Template" on page 7-25

### More About

- "Report Templates" on page 7-2
- "Default Template Contents" on page 7-11



## Customize a Microsoft Word Title Page Template

### In this section...

“Create a Custom Template” on page 7-25  
 “Change the Color of a Report Title” on page 7-25  
 “Assign the Template to a Report” on page 7-27  
 “Customize Title Page Content and Layout” on page 7-27

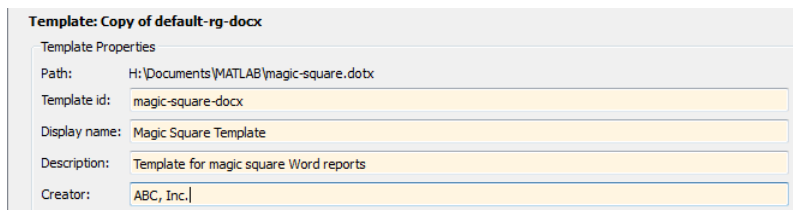
The Report Explorer default Word document conversion template contains document part templates for the front (recto) and back (verso) side of a report title page. The Report Explorer file converter for the Word (from template) output type uses the title page part templates to produce the title pages in the Word output.

This example shows how to create a custom template that changes the color of the title and how to customize the layout of a title page. The example uses a custom template with the Report Generator magic square report example.

### Create a Custom Template

**Note** To complete the rest of this example, you need a custom Word conversion template. If you have a custom template that you want to use for this example, you can skip to “Change the Color of a Report Title” on page 7-25.

- 1 In the Report Explorer, select **Tools > Edit Document Conversion Template**.
- 2 In the list of templates, select the **Default Word Template**.
- 3 In the Template Browser, click **Copy template**.
- 4 In the file browser, navigate to the folder on the MATLAB path that you want to use for the custom template. For the file name, enter `magic-square` and click **Save**.
- 5 In the list of templates, select **Copy of Default Word Template**.
- 6 At the top of the Template Properties dialog box, use these settings:



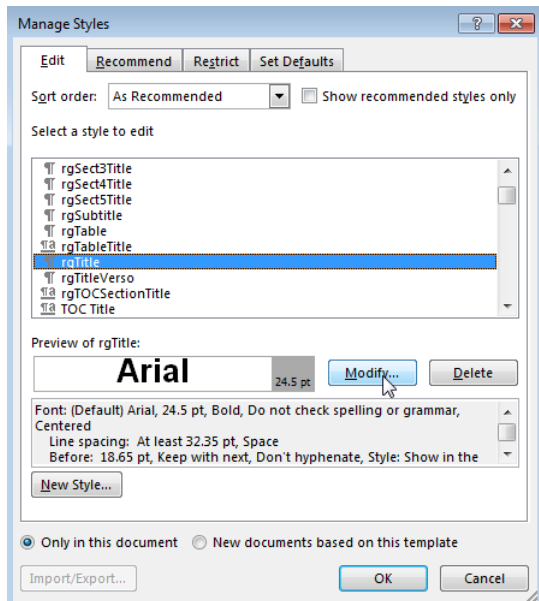
- 7 Apply the properties by selecting another template in the list of templates.

### Change the Color of a Report Title

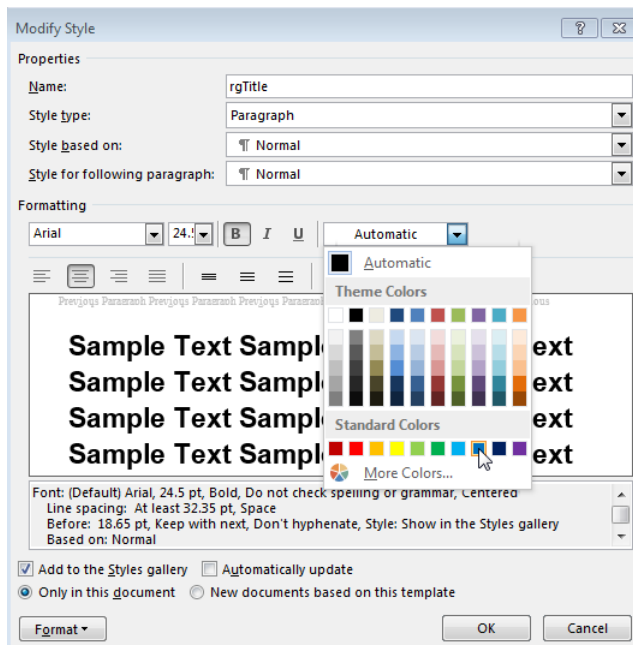
You can customize the Magic Square Template (see “Create a Custom Template” on page 7-25) to use blue text for the report title.

- 1 In the Report Explorer, select **Tools > Edit Document Conversion Template**.

- 2 In the Report Explorer list of Word templates, select Magic Square Template.
- 3 In the Template Browser, click **Open style sheet**. In Word, the template opens, with the Manage Styles dialog box displayed.
- 4 In the Manage Styles dialog box, select the rgTitle style and click **Modify**.



- 5 In the Modify Style dialog box for rgTitle, click the down arrow for Automatic. Select the blue color box and click **OK**.

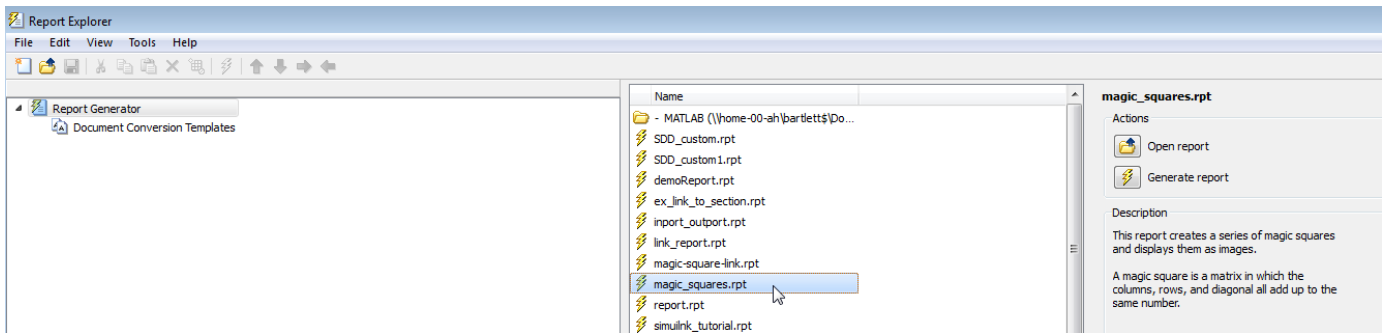


- 6 In the Manage Styles dialog box, click **OK**.
- 7 Save and close the template.

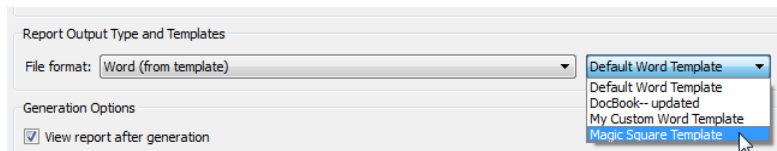
## Assign the Template to a Report


You can assign the customized template to the magic-square.rpt Report Explorer report.

- 1 In the Report Explorer, select **Report Generator** node.
- 2 In the Report Explorer, in the list of reports, select magic\_square.rpt.



- 3 In the Report Options pane, click **Open report**.
- 4 In the magic\_squares report, add a **Title Page** component. In the Title Page dialog box, set the **Title** field to Magic Squares.
- 5 Below the **Title Page** component, add a **Chapter** component.
- 6 In the Report Options dialog box, set **File format** to Word (from template) and instead of Default Word Template, select Magic Squares Template.



- 7 Generate the report. Select the magic\_squares report. In the Report Explorer toolbar, click the **Report**  button.

In the generated report, the title, Magic Squares, appears in blue.

## Customize Title Page Content and Layout

This example assumes you have created a custom Magic Square Template (see “Create a Custom Template” on page 7-25). You can use a different custom Word template.

- 1 In the Report Explorer, select **Tools > Edit Document Conversion Template**.
- 2 In the Report Explorer list of Word templates, select Magic Square Template. In the Report Options pane, click **Open template**.
- 3 With the cursor in the first (and only visible) paragraph in the template, in the **Insert** tab, select the **Quick Parts** button.
- 4 In the Quick Parts gallery, select rgRectoTitlePage to insert of the front title page part template in the main document conversion template.

---

**Tip** To display Word markup for the part template, on the Word ribbon, in the **Developer** tab, click **Design Mode**.

---

- 5 Highlight the rgImage hole and drag it above the rgTitle hole.
- 6 Delete the rgAuthor hole.
- 7 Select the rgRectoTitlePage part template and click the **Quick Parts** button.
- 8 Click **Save Selection to Quick Part Gallery**.
- 9 In the Create New Building Block dialog box, set **Name** to rgRectoTitlePage and the **Category** to mlreportgen. Click **OK**.
- 10 In the template, select the contents of the part template (including the section break) and click the **Delete** button.
- 11 Save and close the template.

Suppose that you use the custom template to generate a report that has a **Title Page** component that specifies an image and an author. The generated report displays the image at the top of the title page and does not include an author.

### See Also

### Related Examples

- “Copy a Template” on page 7-7
- “Customize Microsoft Word Report Styles” on page 7-16

### More About

- “Report Templates” on page 7-2
- “Default Template Contents” on page 7-11

## Create a Custom HTML or PDF Template

### In this section...

“Copy the Template” on page 7-29  
 “Edit HTML or PDF Templates” on page 7-29  
 “Edit HTML or PDF Styles in a Template” on page 7-30  
 “Assign the Template to a Report” on page 7-31  
 “Select an HTML Editor” on page 7-31

To customize the format styles used in the default HTML, single-file HTML, or PDF template, copy the template and modify or add style definitions in the copy.

### Copy the Template

You can copy a default template to use as the basis for your custom template. You can copy a default template in Report Explorer. Alternatively, you can use `mlreportgen.dom.Document.createTemplate` to create a template programmatically.

- 1 In Report Explorer, select **Tools > Edit Document Conversion Template**.
- 2 In the Library pane, select the template you want to copy. For example, select the **Default HTML Template**.
- 3 In the Properties pane, click **Copy template**.
- 4 In the file browser, navigate to where you want to save the template file.

Select a path that is on the MATLAB path, for example, in the MATLAB folder in your home folder.

Specify the file name, using the default file extension for an HTML template (`.html`), single-file HTML template (`.html`), or PDF template (`.pdf`). Click **Save**.

- 5 From the list of templates, select the template copy.
- 6 In the dialog box, in the **Template id** and **Display name** boxes, specify a unique ID and display name for the template.

The display name is the name that appears in the Report Explorer list of templates. Use the template ID to identify a template in your code. See “Set a Template’s Properties” on page 7-8.

- 7 To save the template properties that you entered, click outside of the Properties pane.

### Edit HTML or PDF Templates

A template consists of the main part (`root.html`), which defines the default page, and the document part templates (`docpart_templates.html`). For a single-file HTML template, all parts of the template are in a single file with an `.html` extension. For an HTML or PDF template, the parts of the template are separate files that are packaged into one file with an extension of `.html` or `.pdf`, respectively. When you edit an HTML or PDF template in Report Explorer, Report Explorer unzips the template. After you edit and save the template files, you must package the template files into a single template file by using `zipTemplate`.

You can make similar types of changes in HTML and PDF templates as you can in Word templates. See “Customize Microsoft Word Component Templates” on page 7-18.

The HTML and PDF templates in Report Explorer are similar, with these exceptions:

- PDF templates define a page layout, including page headers and footers. You can modify the document part templates for these layout elements. PDF templates can use a set of DOM API HTML tags supplied for this purpose. See “DOM API HTML Elements” on page 13-33.
- PDF templates can use only a subset of standard HTML elements. See “Standard HTML Elements” on page 13-35.

HTML and PDF templates use DOM API HTML tags to define a document part library and the document part templates within them. The `<dplibrary>` element defines the library. Your template can contain only one `<dplibrary>` tag, which is in place in the default template. The `<dptemplate>` element defines a document part. It takes an argument for the name. For example:

```
<dptemplate name="rgChapter">
```

Look in the `docpart_templates.html` file in your template for some examples.

To edit a template:

- 1 From the list of templates in the middle pane, select the template that you want to edit.

---

**Tip** If the Report Explorer middle pane does not show a list of templates, then select **Tools > Edit Document Conversion Template**.

---

- 2 In the right pane, click **Open template**.
- 3 In the HTML editor, edit and save the template.
- 4 If you edited an HTML or PDF template, package the template files into a single template file by using the `zipTemplate` function at the MATLAB command line.

## Edit HTML or PDF Styles in a Template

You can customize or add format styles in your HTML or PDF template. You edit the styles using cascading style sheets (CSS).

For HTML templates, you can use any CSS property or selector. For PDF, you can use a subset. See “PDF Style Sheets” on page 5-6. You can also use XSL formatting objects (FO) to format elements in a PDF template. However, to simplify and streamline your code, use FO only for properties you cannot define using CSS.

- 1 From the list of templates in the middle pane, select the template that you want to edit.

---

**Tip** If the Report Explorer middle pane does not show a list of templates, then select **Tools > Edit Document Conversion Template**.

---

- 2 In the right pane, click **Open style sheet**.
- 3 In the HTML editor, edit the CSS.

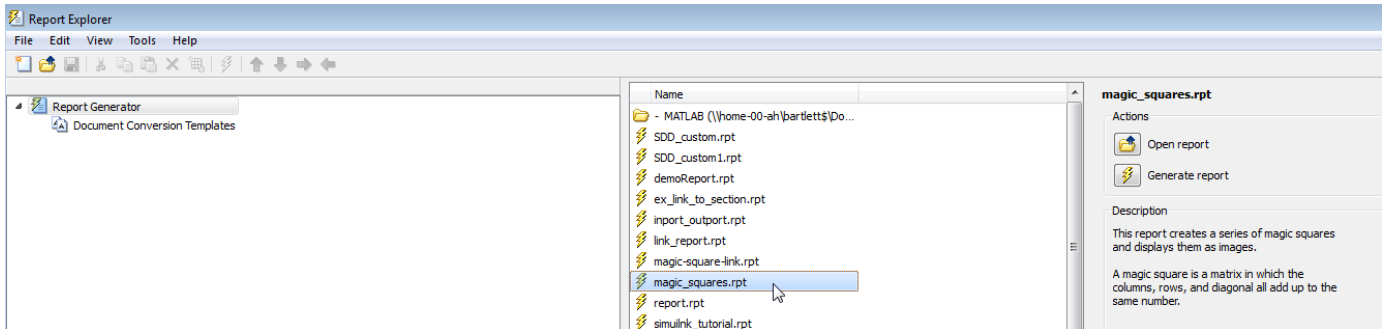
For information about editing a cascading style sheet, see documentation such as the W3Schools.com CSS tutorial.

- 4 Save the style sheet.
- 5 If you edited an HTML or PDF template, package the template files into a single template file by using the `zipTemplate` function at the MATLAB command line.

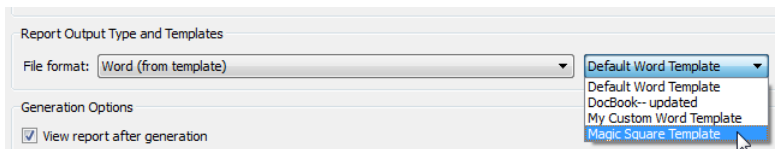
## Assign the Template to a Report

You can assign your template to a Report Explorer report.

- 1 In the Report Explorer, select **Report Generator** node.
- 2 From the list of reports, select the report you want to assign the template to.



- 3 In the Report Options dialog box, set **File format** to one of the (from template) options. Select your template from the list.



## Select an HTML Editor

By default, when you edit an HTML or PDF style sheet, the style sheet appears in the MATLAB Editor.

To use a different editor:

- 1 In the Report Explorer, select **File > Preferences**.
- 2 In **Edit HTML Command**, enter a MATLAB expression that opens the HTML editor you want to use. For example:

```
system('Dreamweaver %<FileName> &')
```

When you open an HTML style sheet, the Report Explorer replaces **FileName** with the template that you selected. The ampersand (&) opens the editor in the background.

## See Also

`mlreportgen.dom.Document.createTemplate`

## Related Examples

- “Generate a Report Using a Template” on page 7-10
- “Customize Microsoft Word Report Styles” on page 7-16
- “Customize Microsoft Word Component Templates” on page 7-18

### **More About**

- “Report Templates” on page 7-2

### **External Websites**

- FO Summary



# Create Custom Components

---

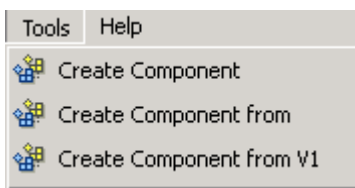
- “Create Custom Components” on page 8-2
- “Define Components” on page 8-4
- “Specify Tasks for a Component to Perform” on page 8-10
- “Define Report Variables” on page 8-15

## Create Custom Components

You can create custom components if you want to generate a report that uses functionality that is not available by using the supplied MATLAB Report Generator components. Custom components help you to generate complex content in your report using a single component. For example, you can create a custom component that builds a table and inserts content into the table based on criteria that you program.

After you complete the process of creating a custom component, you can use the component in the report setup file as you can any component.

- 1 Open the Report Explorer.
- 2 Select one of the component creation choices from the **Tools** menu:



- To create a custom component, select **Create Component**.
- To create a custom component from an existing component, select **Create Component from**.
- To create a component from an existing version 1 component, select **Create Component from V1**.

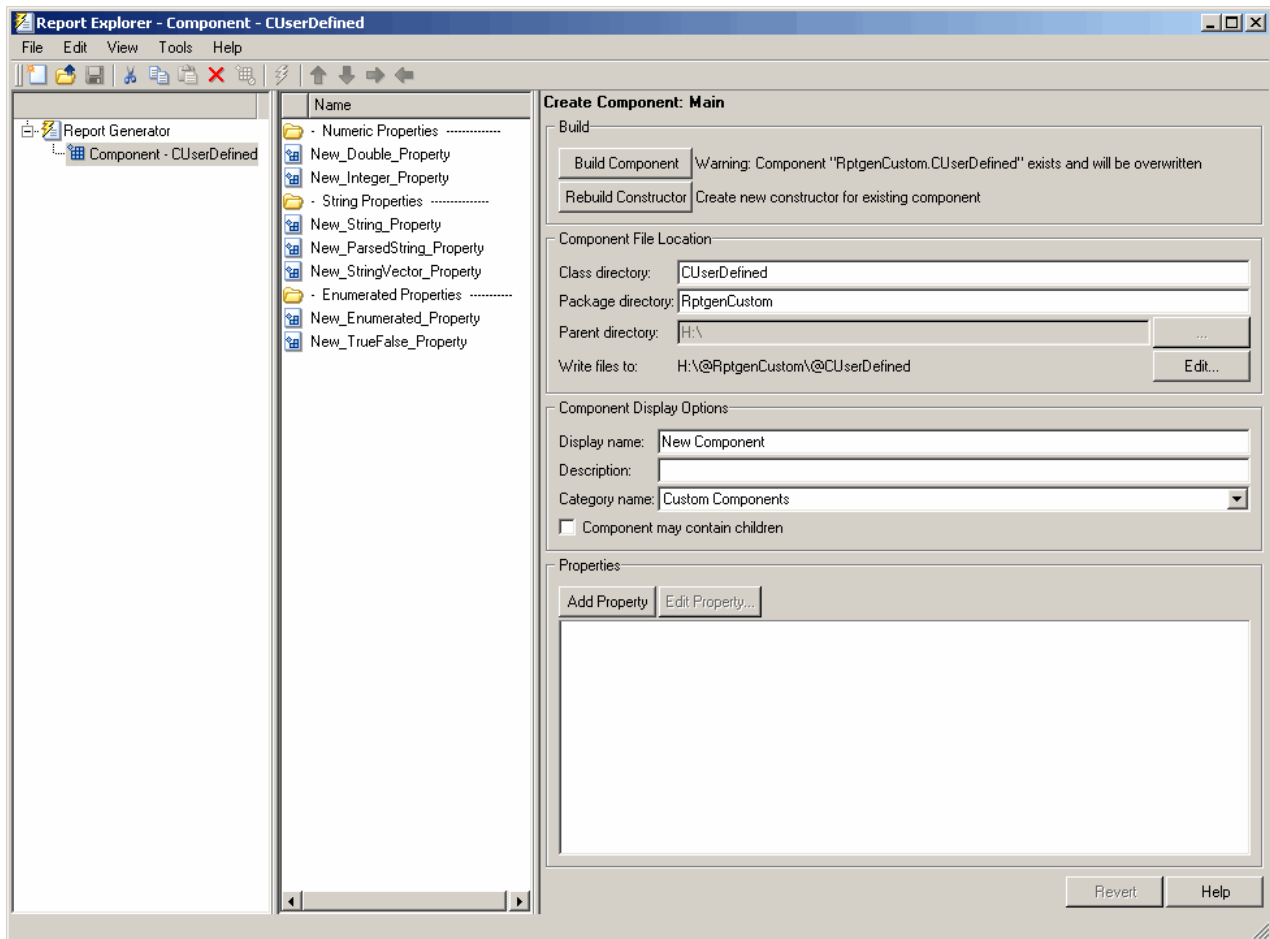
---

**Tip** You can also create a custom component by clicking the **Create a new user-defined reporting component** link in the Report Explorer Properties pane on the right.

---

The Report Explorer displays as follows:

- The Outline pane on the left displays the structure of components you create.
- The Options pane in the middle lists properties you add to components.
- The Properties pane on the right specifies the behavior of component properties.



- 3 Specify properties of the component in the Properties pane of the Report Explorer.
- 4 Specify tasks you want the component to perform by editing the MATLAB files that comprise the framework of the component.
- 5 Build the component.

---

**Note** You must restart the MATLAB session before using a new or rebuilt component.

---

## See Also

### Related Examples

- “Build Components” on page 8-8
- “Specify Tasks for a Component to Perform” on page 8-10
- “Define Components” on page 8-4

## Define Components

### In this section...

“Required Component Data” on page 8-4  
 “Specify the Location of Component Files” on page 8-4  
 “Set Component Display Options” on page 8-5  
 “Specify Component Properties” on page 8-6  
 “Modify Existing Components” on page 8-8  
 “Build Components” on page 8-8  
 “Rebuild Existing Components” on page 8-8  
 “Remove a Component” on page 8-8

### Required Component Data

You must specify the following information when you create a component:

- 1 The path where you want to put the folder that contains all files for the component. For information on how to specify this folder, see “Specify the Location of Component Files” on page 8-4.
- 2 Properties of the component. For more information, see “Specify Component Properties” on page 8-6.
- 3 Display options for the component, including its display name, category, and description. For more information, see “Set Component Display Options” on page 8-5.

### Specify the Location of Component Files

You can create components that perform similar functions and group them in *Package Directories*. Each package folder must have a *Parent Directory* that is in the MATLAB path. When you build a new component, the MATLAB Report Generator software creates files that make up the component. These files are stored in the folder structure <parent>/@package\_name/@class\_name.

Specify these directories in the following fields in the **Component File Location** area of the Properties pane:

- 1 **Class Directory** field. Specify a class name for your component. The build process creates a folder with the name you specify and places the component's files in it. The class folder name must be unique for each component in the package. By convention, component class names begin with an uppercase or lowercase letter *c*; for example, *cUserDefinedComponent*.
- 2 **Package Directory** field. Specify the folder in which to store files for groups of components you create. Files for each component are stored in a subfolder with the name you entered into **Class Directory Field**.
- 3 **Parent Directory** field. Specify this folder when you create a package for the first time. This folder is the parent folder of the Package Directory.

## Set Component Display Options

You can specify how you want your component to appear in the Report Explorer by entering data in the **Component Display Options** area of the Properties pane. Enter the following information:

- 1 **Display Name.** Specify a display name for the component to appear in the list of components for its associated category. Component categories and display names appear in the Options pane in the middle of the Report Explorer.

For information on specifying component categories, see step 3, **Category Name**.

The following example shows how to create a component called My First Component in a category called My Category.

Component Display Options

Display name: My First Component

Description: The first component

Category name: My Category

Component may contain children

- 2 **Description.** Enter a description for the component. This description appears when you click the component name or category name in the Options pane in the middle of the Report Explorer. Make the description informative, but brief.
- 3 **Category Name.** Specify the category of components to which the new component belongs. The component appears under this category in the Options pane in the middle of the Report Explorer.

Predefined choices appear in the **Category name** list. Select a component category from this list.

Component Display Options

Display name: Example String Component

Description: A MATLAB String

Category name: MATLAB

Component may contain children

Properties

Add Property

- Formatting
- Handle Graphics
- Logical and Flow Control
- MATLAB**
- Real-Time Workshop
- Report Generator
- Requirements Toolbox
- Simulink
- Simulink Blocks
- Simulink Fixed Point

To create a custom component category, type the name for the category into the **Category name** field. This category name appears in the list of available categories in the Report Explorer.

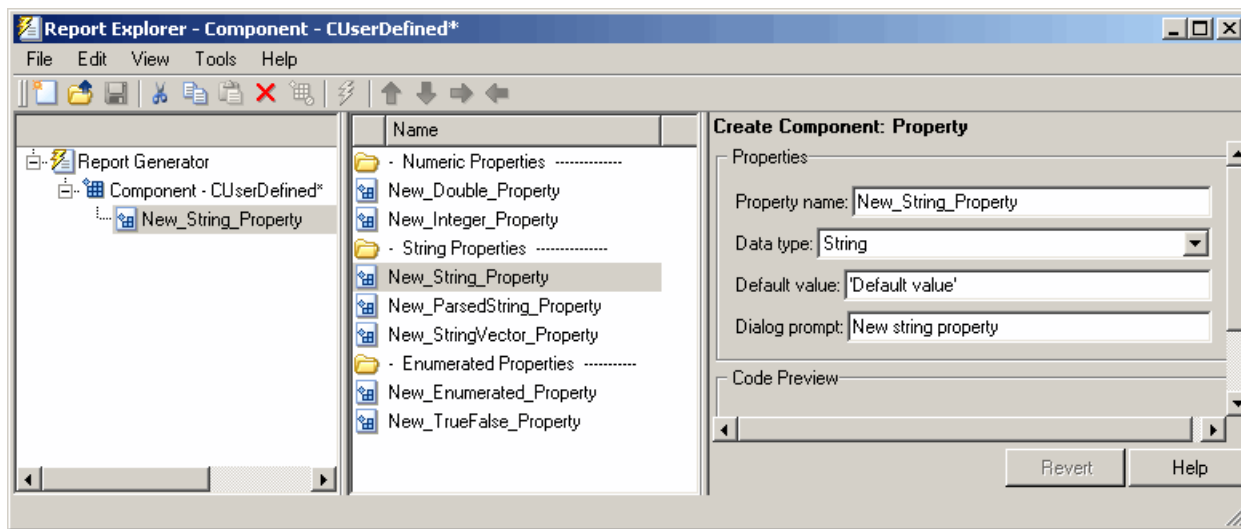


#### 4 child components.

Select the **Component may contain children** check box if you want the component to have child components. Child components appear under the component in the Report Explorer hierarchy. During report generation, the component runs all child components and includes their output in the report.

## Specify Component Properties

Component properties determine how a component behaves and what information it inserts into a report. To see the current value of a component's property, double-click it in the Outline pane on the left in the Report Explorer. For example, the figure shows the property values for `New_String_Property`.

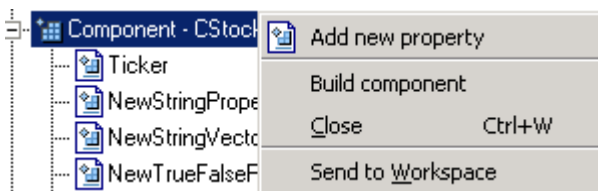


### Add Properties to Components

You add properties to a component from the properties list. Each property has a default value that you can modify as needed.

There are several ways to add properties to components:

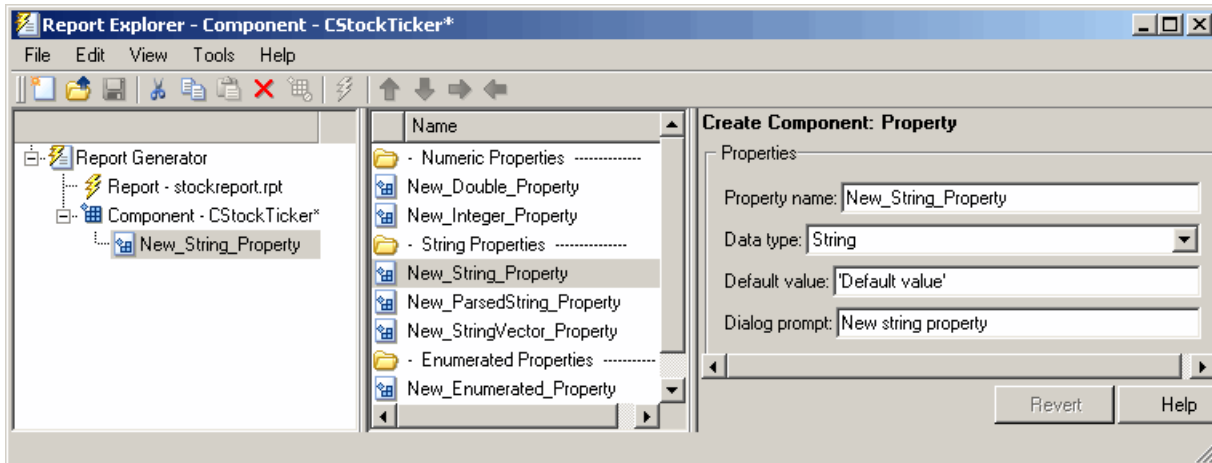
- 1 Right-click the name of the component to which you want to add properties in the Outline pane on the left. Select **Add new property** from its context menu.



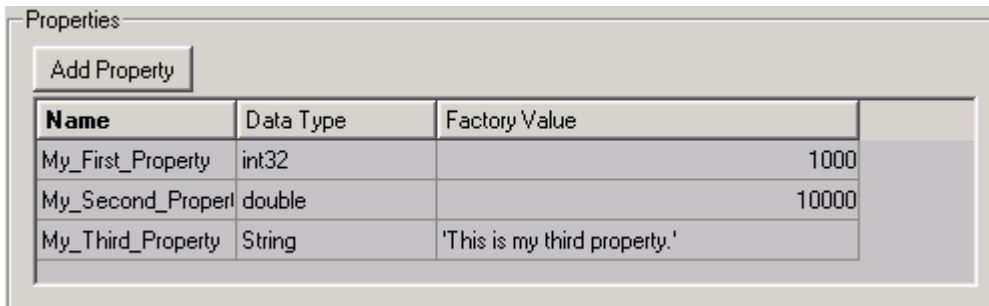
- 2 Right-click the name of a predefined property in the Options pane in the middle. From the context menu, select **Add property**.



- 3 Left-click the name of a property in the Options pane, and then drag it on top of a component in the Outline pane on the left.
- 4 Double-click the property name in the Options pane in the middle. The property is added to the component and property values appear in the Properties pane on the right.



- 5 Click the **Add Property** button on the Properties pane on the right.



### Specify Component Properties

- 1 **Property Name.** Create a name for the new property. A property name must be a valid MATLAB variable name, and must be unique within a component.
- 2 **Data Type.** Specify the property's data type. Options are:
  - Double
  - Enumeration
  - Integer
  - Character vector
  - Character-vector vector
  - %<ParsedString>

Use this data type to include the value of a variable in the MATLAB workspace in a component.

- True/False
- 3 Default Value.** Set a default value for the property. The default value must be compatible with the data type. If incompatibilities exist between the default value and the data type, the component might not build.
  - 4 Dialog Prompt.** This text appears next to the widget on the component's dialog box. It indicates what the property does and how it affects report generation.

---

**Note** When the component builds, a colon is appended to your entry in the **Dialog prompt** field. Your entry appears in the Properties pane with the colon appended.

---

## Modify Existing Components

Report components are modifiable. You can derive a new component from an existing component by double-clicking the name of the component and modifying its values and properties.

## Build Components

After you have entered all data required for defining the component, you build it by clicking the **Build Component** button. The build process creates all files needed for the component and stores them in the specified folder. For more information about specifying where components are stored, see “Specify the Location of Component Files” on page 8-4.

---

**Note** Existing files in this location are overwritten.

---

## Rebuild Existing Components

To add, remove, or change properties of an existing component, use the **Rebuild Constructor** button. This button becomes active only after you have previously created a component using the **Build Component** button. To activate the **Rebuild Constructor** button, specify the **Package name** and **Class name** for an existing component. These fields are located in the **Component File Location** area of the Properties pane.

If you select a component using **Tools > Create component from**, the component's fields are filled in automatically and the button becomes active.

After you have finished modifying the component, click the **Rebuild Constructor** button to rebuild the component. Writable files in the component's folder location are not overwritten.

## Remove a Component

To remove a component:

- 1** Delete its class folder, <root>/@package\_name/@class\_name. If the component you want to remove is the only component in the package, delete the entire package.



- 2 Edit `<root>/@package_name/rptcomps2.xml` to remove the XML element that registers the component.

## Specify Tasks for a Component to Perform

### In this section...

“About Component Customization” on page 8-10

“Required Customization: Specify Format and Content of Report Output” on page 8-10

“Change a Component's Outline Text in the Report Explorer Hierarchy” on page 8-12

“Modify the Appearance of Properties Dialog Boxes” on page 8-12

“Specify Additional Component Properties” on page 8-13

### About Component Customization

Building a component creates MATLAB files in the MATLAB workspace. Specify tasks that you want your component to perform by editing these MATLAB files.

**Note** You *must* specify the format and content of your report output by editing `execute.m`. This file is called during report generation to invoke your component's tasks. Optionally, you can specify additional component properties and behavior by editing other MATLAB files.

For more information, see the following sections:

- “Required Customization: Specify Format and Content of Report Output” on page 8-10
- “Change a Component's Outline Text in the Report Explorer Hierarchy” on page 8-12
- “Modify the Appearance of Properties Dialog Boxes” on page 8-12
- “Specify Additional Component Properties” on page 8-13

### Required Customization: Specify Format and Content of Report Output

After you build the component, specify the format and content of your report output by editing the `execute.m` file.

The `execute` command has the following syntax:

```
out = execute(thisComp, parentDoc)
```

Where:

- `thisComp` is a handle to the component that you are running.
- `parentDoc` is a handle to the document that you are generating.
- `out` is a Document Object Model (DOM) node or string to add to the report.

For information on manipulating DOM nodes, see `xmlwrite` in the MATLAB documentation.

One or more default lines of code within the `execute.m` file show each property for the component. Here is an example of a component property line within an `execute.m` file:

```
pstring = thisComp.NewStringProperty; % New string property;
```

The following sections describe how to edit `execute.m` to create additional report elements.

**Create Tables**

To create a table, replace the Source property value with the name of a cell array or structure:

```
out = execute(rptgen.cfr_table(...
'Source', tableSrc,...
'numHeaderRows',1,...
'TableTitle','Example Title'),...
parentDoc);
```

For more information, enter `help(rptgen.cfr_table)` at the MATLAB command line.

**Create Lists**

To create a list, replace the Source property value with the name of a cell vector:

```
out = execute(rptgen.cfr_list(...
'Source', listSrc,...
'ListStyle','orderedlist',...
'ListTitle','Example List'),...
parentDoc);
```

For more information, enter `help(rptgen.cfr_list)` at the MATLAB command line.

**Create Text**

To create text, replace the ParaText property value with a character vector:

```
out = execute(rptgen.cfr_paragraph(...
'ParaText', paraSrc,...
parentDoc);
```

For more information, enter `help(rptgen.cfr_paragraph)` at the command line.

**Create Figures**

To create figures, specify a figure in the FigureHandle property value.

```
figSrc =(gcf;
out = execute(rptgen_hg.chg_fig_snap(...
'FigureHandle', figSrc,...
'Title', '',...
'isResizeFigure', 'manual',...
'PrintSize', [6 4],...
'PrintUnits', 'inches'),...
parentDoc);
```

For more information, enter `help(rptgen_hg.chg_fig_snap)` at the MATLAB command line.

**Run Child Components**

The following code runs child components. The first line calls `execute.m` for child components. The second line appends the results of running the child components to the report:

```
childOut = thisComp.runChildren(parentDoc);
out = parentDoc.createDocumentFragment(out, childOut);
```

## Change a Component's Outline Text in the Report Explorer Hierarchy

To change the string used to describe the component in the Report Explorer hierarchy, edit the `getOutlineString` MATLAB file. By default, `getoutlinestring` returns the display name of the component. The `getOutlineString` command has the following syntax:

```
olstring = getOutlineString(thisComp)
```

Where:

- `thisComp` is the component whose description you are specifying.
- `olstring` is a single-line that displays information about the component. It can contain a maximum of 32 characters.

Customize the string to include additional information about the component, such as information about its properties. In the following example, the `truncatestring` function converts input data into a single-line character vector. If the data is empty, the second argument is the return value, The third argument is the maximum allowed size of the resulting character vector.

```
cInfo = '';  
pstring = rptgen.truncateString(thisComp.string, '<empty>', 16);
```

Use a dash (-) as a separator between the name and additional component information, as follows:

```
if ~isempty(cInfo)  
    olstring = [olstring, '-', cInfo];  
end
```

## Modify the Appearance of Properties Dialog Boxes

You can edit the `getdialogschema.m` file to control most aspects of dialog box layout, including:

- Creation and placement of widgets
- Organization of widgets into panes
- Creation of the top-level display within which panes reside

The syntax of the command is:

```
dlgstruct = getdialogschema(thisComp, name)
```

Where:

- `thisComp` is the instance of the component being edited.
- `name` is a character vector that is passed to `getdialogschema` to build a specific type of pane. Usually, `name` is empty in the Report Explorer.

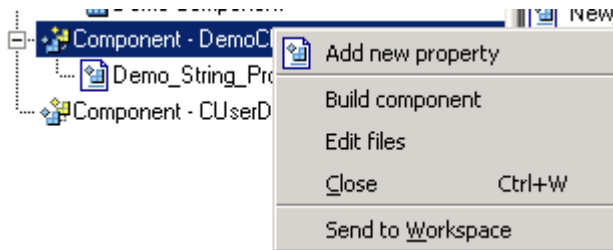
---

**Note** Do not modify fields that are not explicitly included in this file. These fields are subject to change in future releases.

---

## Specify Additional Component Properties

You can edit additional MATLAB files to customize your component further. To access these files, right-click the component in the Outline pane on the left in the Report Explorer and select **Edit files** from its context menu.



For more information, see the following sections:

- “Specify Whether Components Can Have Children Components” on page 8-13
- “Modify a Component Description” on page 8-13
- “Change a Component Display Name” on page 8-13
- “Change a Component Category Name” on page 8-13
- “Register Components” on page 8-14
- “Display Component Help in the MATLAB Help Browser” on page 8-14

### Specify Whether Components Can Have Children Components

To specify whether a component can have children, edit `getParentable.m`. This command returns the value `true` or `false`. For example, if you no longer want your component to have child components, modify the value within the code as follows:

```
p = false;
```

### Modify a Component Description

The description in `getDescription.m` is the same value as the **Description** field in the Report Explorer. The following example shows how to edit the `compDesc` value in this file to change a component's description to `An example component`:

```
compDesc = 'An example component';
```

### Change a Component Display Name

The display name in `getName.m` is the same value as the **Display name** field in the Report Explorer. The following example shows how to edit the `compName` value in this file to change a component's display name to `Example Component`:

```
compName = 'Example Component';
```

### Change a Component Category Name

The category name in `getType.m` is the same value as the **Category name** field in the Report Explorer. The following example shows how to edit the `compCategory` value in this file to change a component's category name to `Custom Components`:

```
compCategory = 'Custom Components';
```

### **Register Components**

You can register components in the Report Explorer using `rptcomps2.xml`. This file also helps build the list of available components.

The content of this file must be consistent with the values in the `getName.m` and `getType.m` files. If you have changed values in either of these files, you must also change their values in `rptcomps2.xml`. You must restart the MATLAB software session for the Report Explorer to display new information.

### **Display Component Help in the MATLAB Help Browser**

The `viewHelp.m` file displays a help file for the component within the MATLAB Help browser. To display the help file, highlight the name of the component in the Report Explorer and click **Help**.

## Define Report Variables

You can have your report use variables defined in the MATLAB workspace to specify values to be used by components. You can create these variables in the MATLAB workspace before running the report. However, a better solution is to let the report create the variables, using the Evaluate MATLAB Expression component.

For an example, see “Specify Report Variables” on page 2-12.





# Create Custom Style Sheets

---

- “Style Sheets” on page 9-2
- “Create a New Style Sheet” on page 9-4
- “Edit, Save, or Delete a Style Sheet” on page 9-5
- “Edit Style Sheet Data Items” on page 9-8
- “Style Sheet Cells for Headers and Footers” on page 9-19
- “Customized Style Sheets” on page 9-23
- “Configure PDF Fonts” on page 9-32

## Style Sheets

### In this section...

“Built-In Versus Custom Style Sheets” on page 9-2

“Customize Style Sheets Using Data Items” on page 9-2

### Built-In Versus Custom Style Sheets

*Style Sheets* specify formatting and display settings for reports. The report-generation process uses style sheets to convert reports from DocBook XML format to a format that you specify. If you want to generate the given report in a different format than initially specified, you can convert the XML document using a different or modified style sheet.

The following table lists report output formats and their default style sheets.

Report Format	Default Style Sheet
HTML	Uses style sheets for either single- or multiple-page documents
PDF	Formatting Object (FO) style sheet
RTF, Word	Document Style Semantics and Specification Language (DSSSL) style sheet

The following table shows a list of properties for the built-in style sheets.

### Properties of Style Sheets

Name	Description
Description	A description of the style sheet.
Display name	The style sheet name that appears in the Options pane.
Transform type	<p>The process used to generate reports that use a specified style sheet. Supported types are:</p> <ul style="list-style-type: none"> <li>HTML</li> <li>FO (Formatting Object) for PDF reports</li> <li>DSSSL (Document Style Semantics and Specification Language) for RTF and Word reports</li> </ul>
	<b>Note</b> This field is not editable.

In most cases, the style sheets provided with the MATLAB Report Generator software should be more than adequate for your needs. However, you may want to modify the built-in style sheets to meet special requirements. For example, suppose one of the built-in style sheets meets your requirements, but you want to change the page orientation. You can create a custom style sheet by editing the built-in style sheet to your specifications.

### Customize Style Sheets Using Data Items

Each built-in style sheet includes editable styles, also called *data items*, organized in categories. These data items specify styles that the file converter uses for a given report. You can edit these data items to customize style sheets for your reports.

Data items can be of different types, some of which require different editing methods. For more information about editing data items, see “Edit Style Sheet Data Items” on page 9-8.

---

**Tip** See the **Help** area at the bottom of the Properties pane on the right for a description of a specific data item that you are editing.

---

## Create a New Style Sheet

To create a style sheet:

- 1 Open the Report Explorer.
- 2 From the menu bar, click **Tools > Edit Style Sheet**.
- 3 In the Properties pane on the right, choose the built-in style sheet for the format with which you want to work. Options are:
  - **New HTML**. Creates a style sheet for HTML reports.
  - **New multi-page HTML**. Creates a style sheet for HTML reports with more than one page.
  - **New FO (PDF)**. Creates a style sheet for PDF reports.
  - **New DSSSL (RTF)**. Creates a style sheet for RTF reports.

The new style sheet appears in the Outline pane on the left.

- 4 In the Properties pane on the right, modify the properties for the style sheet as needed. Add data items to the new style sheet:
  - a Drag the data item you want to add from the Options pane in the middle to the style sheet in the Outline pane on the left.
  - b In the Properties pane on the right, edit the data items for the selected style. For more information, see “Edit Style Sheet Data Items” on page 9-8
- 5 Save the style sheet. For information about how to save a style sheet, see “Save a Style Sheet” on page 9-7.

## Edit, Save, or Delete a Style Sheet

In this section...
"Edit a Style Sheet" on page 9-5
"Save a Style Sheet" on page 9-7
"Delete a Style Sheet" on page 9-7

### Edit a Style Sheet

To edit a style sheet:

- 1 In Report Explorer, select a report setup file in the Outline pane on the left.
- 2 From the menu bar, click **Tools > Edit Style Sheet**.

The Report Explorer displays as follows.

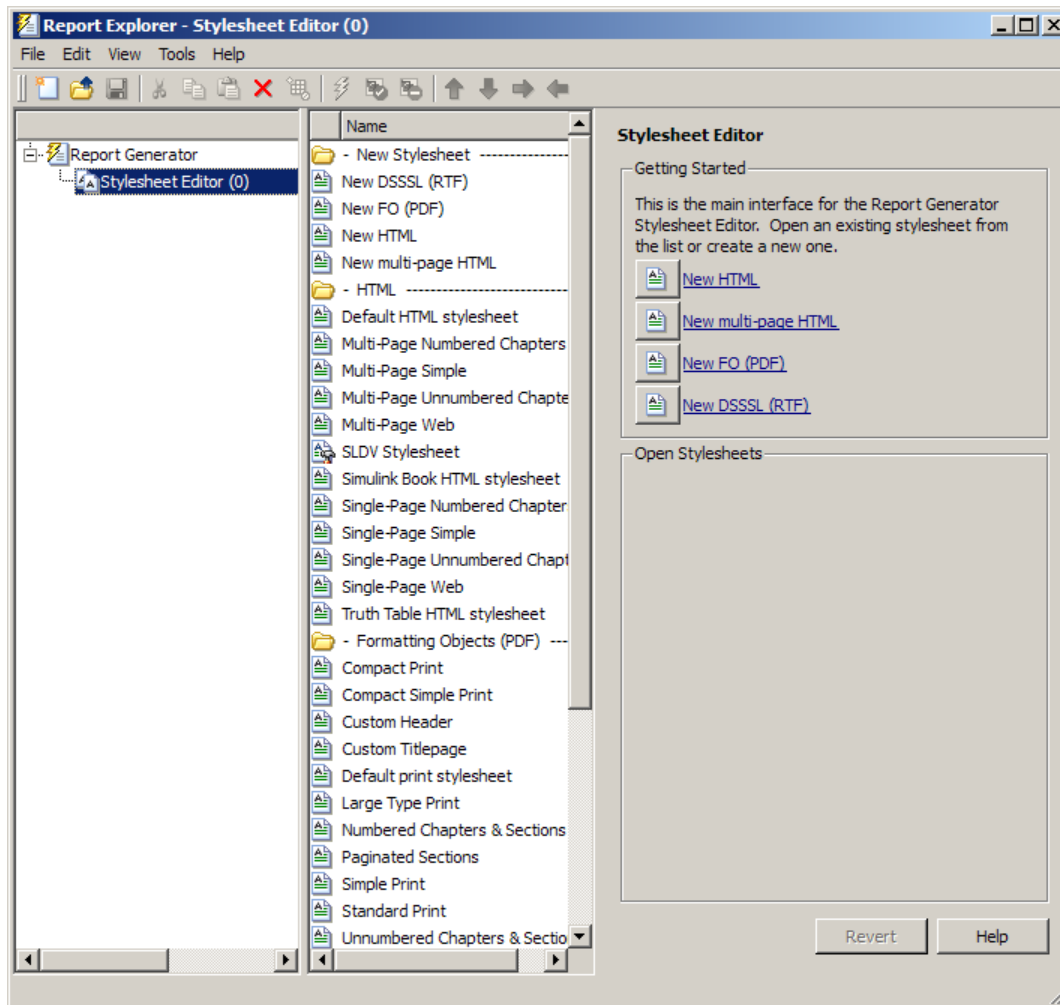
- The Outline pane on the left displays the structure of style sheets you create.
- The Options pane in the middle lists style sheets available for customizing.

---

**Tip** Double-click a category to collapse it. Double-click it again to expand it.

---

- The Properties pane on the right shows properties of style sheets, such as name and description.



You can use the Report Explorer to work with style sheets as follows.

Task	Pane to Use	Action
Create a style sheet	Properties	Click the link that corresponds to the kind of style sheet you want to create
Open an existing style sheet	Properties	Click the name of the style sheet, which appear in the <b>Open Style Sheets</b> area
Select a style sheet to use for converting an XML source file	Options	Select a style sheet by clicking on it
View a list of customized styles in a style sheet	Outline	Expand any open style sheet
View a list of styles in a style sheet	Outline or Options	Double-click the style sheet
View a list of style sheets available for editing in a given category	Options	Double-click the folder that corresponds to the kind of output you want (that is, HTML, PDF, RTF, or Word)

Task	Pane to Use	Action
View open style sheets	Outline	Expand the Style Sheet Editor item in Report Explorer
Change the name or description of the current style sheet	Properties	Edit the text in the <b>Display Name</b> or <b>Description</b> field.
Convert an XML source file using the current style sheet	Properties	Click <b>Send to Source File Converter</b> in the Properties pane.
Edit customized style data	Properties	Click the style data item, which appears in the <b>Style Sheet Customizations</b> area
Open a style data item for editing or viewing	Options	Double-click the data item that you want to edit.
View a list of customized style data	Outline	Expand the style sheet

## Save a Style Sheet

You must save a style sheet before you can use it to convert a source file or associate it with a report. To use the Report Explorer to save a style sheet:

- 1 Select the style sheet that you want to save in the Outline pane on the left.
- 2 Select **File > Save As** from the menu bar and specify a new name for the style sheet (to avoid overwriting built-in style sheets). You must save the file in a folder in your MATLAB path for the style sheet to appear in the Report Explorer. The file name must be unique in the MATLAB path.

By convention, MATLAB Report Generator style sheets have `.rgs` as their file name extension.

## Delete a Style Sheet

To use the Report Explorer to delete a style sheet that you created:

- 1 Select the style sheet that you want to delete in the Outline pane on the left.
- 2 Click the style sheet to delete from the Options pane in the middle.
- 3 Click **Delete style sheet** in the style sheet's Properties pane on the right.

You must restart the MATLAB software session for deleted style sheets to disappear from the Options pane.

---

**Note** You cannot delete built-in style sheets.

---

## Edit Style Sheet Data Items

In this section...
"Data Item Categories in Built-In Style Sheets" on page 9-8
"Edit Data Items in Simple or Advanced Edit Mode" on page 9-11
"Data Items" on page 9-12

### Data Item Categories in Built-In Style Sheets

You can edit data items in built-in style sheets to customize them. Data items appear in *categories*, according to their function. The following tables list the categories and data items for each type of style sheet provided with the MATLAB Report Generator software.



**Categories of Styles in PDF (FO) Style Sheets**

<b>Category</b>	<b>Description of Data Items in Category</b>
Automatic labeling	Options for enumeration of parts of the report, such as chapters and sections
Callouts	Options and specifications related to callouts, such as defaults, use of graphics, size, path, fonts, characters, and extensions
Cross References	Option to control whether page numbers appear in report
Font Families	Specification of defaults for body text, copyright, quotes, symbols, dingbats, monospace, sans serif, and titles
Graphics	Specification of default width and options related to scaling attributes
Lists	Specification of spacing related to lists and list items
Meta/*Info	Options related to year ranges
Miscellaneous	Options and specifications for placement of titles, comments, variable lists, block quotations, ulinks, hyphenations of URLs, verbatim environment display, use of SVG, table footnote numbers, superscript, and subscript
Pagination and General Styles	<p>Specifications of page orientation, margins, double-sided, paper type, hyphenation, line height, columns, master font, draft mode, watermark, blank pages, rules for headers and footers, and content of headers and footers</p> <p><b>Note</b> You can specify parameters in this category, such as margin widths and header and footer height, in units of inches (in), millimeters (mm), or picas (pc), where 1 pica = 1/6 inch.</p>
Property Sets	Specification and options related to figure titles, monospace properties, verbatim text, section titles, and levels of sections
Reference Pages	Option to control whether the class name is displayed
Style Sheet Extensions	Line numbering and table columns extensions
Table of Contents (TOC)/List of Tables (LOT)/Index Generation	Specifications for layout of TOC, depth of sections, indentation, and margins
Tables	Specifications for size of tables and their borders
Title Page	Specifications for positioning and transformation of title page elements and properties of title page text elements

For information about DocBook XSL style sheets, see <http://docbook.sourceforge.net/release/xsl/current/doc/>.

You can set up font mappings for non-English PDF fonts. The PDF style sheets override those mappings. For details, see

**Categories of Styles in HTML and Multi-Page HTML Style Sheets**

<b>Category of Style</b>	<b>Description of Data Items in Category</b>
Automatic labeling	Options for enumeration of parts of the report, such as chapters and sections
Callouts	Options and specifications related to callouts, such as defaults, use of graphics, size, path, fonts, characters, and extensions
Chunking	Options related to using an explicit TOC for chunking, depth of section chunks, navigational graphics, and display of titles in headers and footers
Style Sheet Extensions	Line numbering, graphic size, and table columns extensions
Graphics	Specification of default width and depth, use of HTML embed for SVG, viewports, and options related to scaling attributes
HTML	Specifications related to dynamically served HTML, base and head elements, type of style sheet, css, propagation of styles, longdesc, validation, cleanup, draft mode, watermark, and generation of abstract
Linking	Specification of Mailto URL and target for ulinks
Meta/*Info	Options related to year ranges
Miscellaneous	Options and specifications for comments, verbatim environment pixels, em space, use of SVG, and table footnote numbers
Reference Pages	Option control whether the class name is displayed
Table of Contents (TOC)/List of Tables (LOT)/Index Generation	Specifications for layout of TOC, depth of sections, indentation, and margins
Tables	Specifications for size of tables, table cell spacing and padding, and borders
Title Page	Specifications for positioning and transformation of title page elements and properties of title page text elements
XSLT Processing	Options related to header and footer navigation and rules

For information about:

- DocBook — see <https://www.oasis-open.org/docbook/documentation/reference/html/docbook.html>
- DocBook XSL style sheets — see <http://docbook.sourceforge.net/release/xsl/current/doc/>
- DocBook print parameters, see <http://docbook.sourceforge.net/release/dsssl/1.79/doc/print/>

## Categories of Styles in RTF (DSSSL) Style Sheets

Category of Style	Description of Data Items in Category
Admonitions	Options and path for admonition graphics
Backends	Options for Tex, MIF, and RTF back-end usage
Bibliographies	Options related to checking citations; suppressing, enumerating, and using titles of entries
Fonts	Specifications for font family and size to use for some elements
Footnotes	Options for ulinks as footnotes and page location
Graphics	Specifications for file extensions, file names, and loading library database
Indents	Specifications for hanging indents, first paragraphs, and start of blocks
Labeling	Enumeration of sections and other elements
Miscellaneous	Options for floating formal objects, punctuation for run-in heads and honorifics, bold for first use of term, minimum leading between lines, and automatic hyphenation
OLinks	Using an extension for finding outline information
Object Rules	Specifications for placement and width of rules
Paper/Page Characteristics	Specifications for paper type, page numbers, width of pages, margins, and columns; heading-levels, sides; and writing mode (such as left-to-right)
Quadding	Specifications for justifying paragraphs
RefEntries and Functions	Options related to generation and display of reference entries and synopses for functions
Running heads	Options for generating and displaying running heads of chapters
Table of Contents (TOC)/List of Tables (LOT)	Options to produce or display TOC for sets, books, parts, references, articles. Options to display TOC on title page
Tables	Specification of width in simple list
VariableLists	Options and specifications for term length and formatting
Verbatim Environments	Specifications for width, enumeration, size, indentation, line frequency, and callouts
Vertical Spacing	Specifications for space between lines and paragraphs

## Edit Data Items in Simple or Advanced Edit Mode

- To edit a data item in *simple edit mode*, edit a simple text value that corresponds to the data in the style sheet. This value appears in the field to the right of the **Value** label. For some values, use a selection list to change the value instead of typing in text.
- To edit a data item in *advanced edit mode*, edit the XML code directly.

---

**Note** This section gives instructions for simple edit mode, except where explicitly specified otherwise.

---

The user interface is in simple edit mode when the data item appears in a pane labeled **Value**. It is in advanced edit mode when the data item appears in a pane labeled **Value (XML)**. To switch from simple to advanced edit mode, click **Edit as XML**.

Edit values for most data items in PDF and HTML style sheets in either simple edit mode or advanced edit mode. Edit values for RTF style sheets in simple edit mode only. Data items in RTF style sheets do not support advanced edit mode.

---

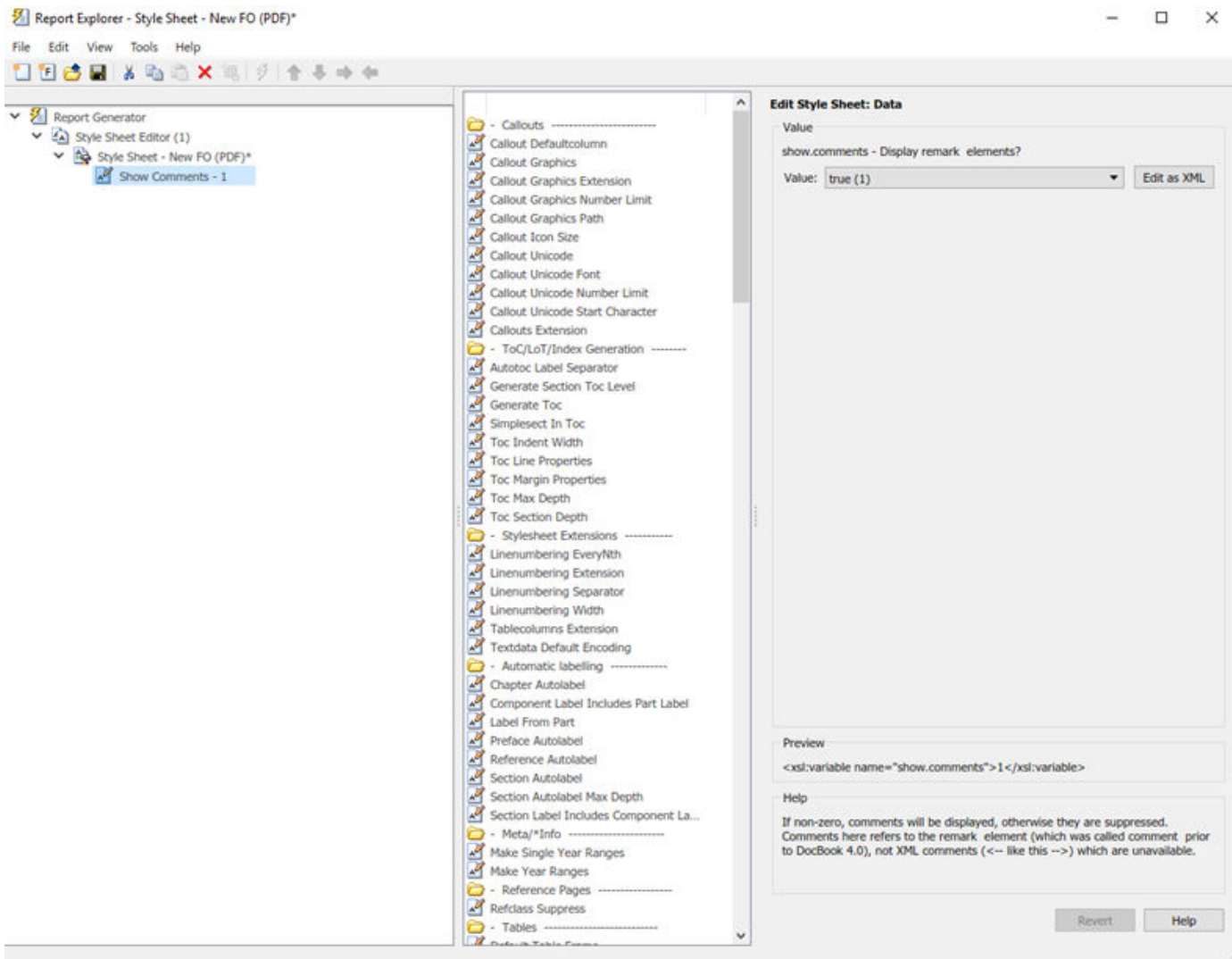
**Note** To modify content for headers and footers you edit *style sheet cells*, which do not appear in either simple or advanced mode. For more information, see “Style Sheet Cells for Headers and Footers” on page 9-19.

---

### Data Items

Select a style sheet from the Options pane in the middle of the Report Explorer. The Outline pane on the left shows the name of the current style data item inside its style sheet. The Options pane in the middle shows a list of available style sheet data items. The Properties pane on the right displays **Edit Style Sheet: Data**. It also includes the following information:

- The value of the data item is in a pane labeled **Value** in simple edit mode or **Value (XML)** in advanced edit mode.
- To the right of the value is the **Edit as XML** toggle button.
- The **Preview** pane includes a partial view of the style sheet that specifies the data item. The data in this pane is not editable.
- The **Help** pane contains information about the data item. This information is not editable.



## Edit Boolean and Enumerated Values

In the previous figure, the `Show Comments` data item is of type `Boolean`. Its current value is `true(1)`. Change this value using the menu list for the value field. In this case, the only other possible value is `false(2)`.

## Edit Values

For the values of some data items, the Report Explorer displays text in the editable **Value** field. You can specify an XML expression, though you are not required to do so.

## Edit XML Expressions

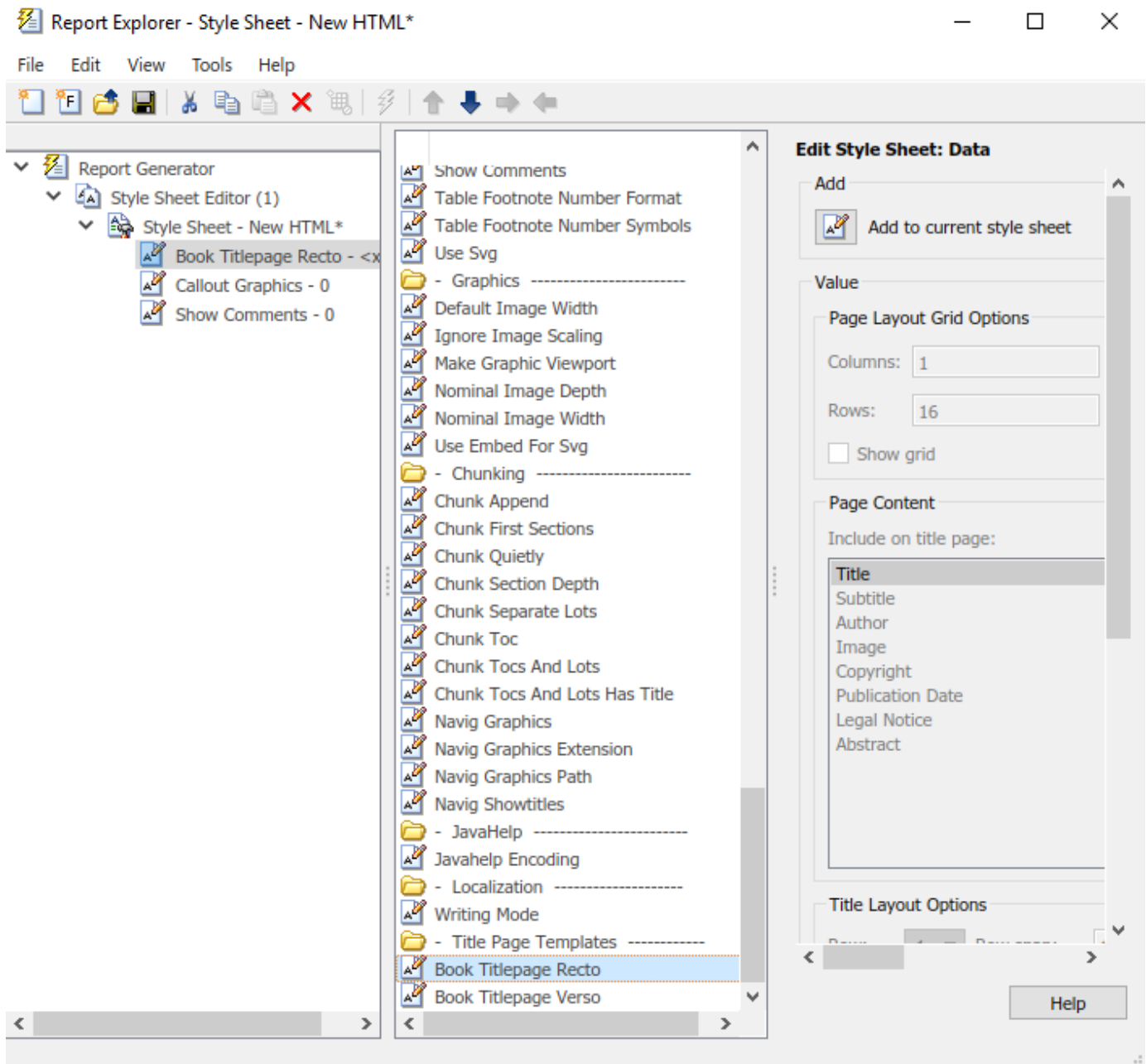
To make complex changes to a style sheet, consider using Advanced edit mode. This enables you to edit XML expressions directly in the **Value (XML)** pane. If this pane does not appear, click **Edit as XML** to switch to advanced edit mode.

Make sure that you enter valid XML. Invalid XML values generate an error, which appears at the top of the Properties pane.

## Modify Title Page Properties

For PDF or HTML style sheets, you can modify the layout, contents, and format of a title page by using the Style Sheet Editor.

- 1 In the Outline pane, select the style sheet you want to edit.
- 2 In the Options pane, in the **Title Page Templates** section, select:
  - **Book Titlepage Recto** to specify properties for the front side of the title page
  - **Book Titlepage Verso** to specify properties for the back side of the title page
- 3 In the Properties pane, select **Add to current style sheet** and edit the properties.



To adjust the grid used to position the title page elements (such as the title and author) on the page, in the Properties pane specify:

- **Columns** — The number of columns in the page grid
- **Width** — The width of each column
- **Rows** — The number of rows in the page grid
- **Width** — The width of each row

To view the grid layout on the generated title page, select **Show grid**.

By default, all of the title page elements appear on the title page. To exclude display of a title page element:

- 1 In the Properties pane, in the **Include on title page** list, select an element to exclude.
- 2 Click the right arrow button. The element appears in the **Exclude from Title Page** list.

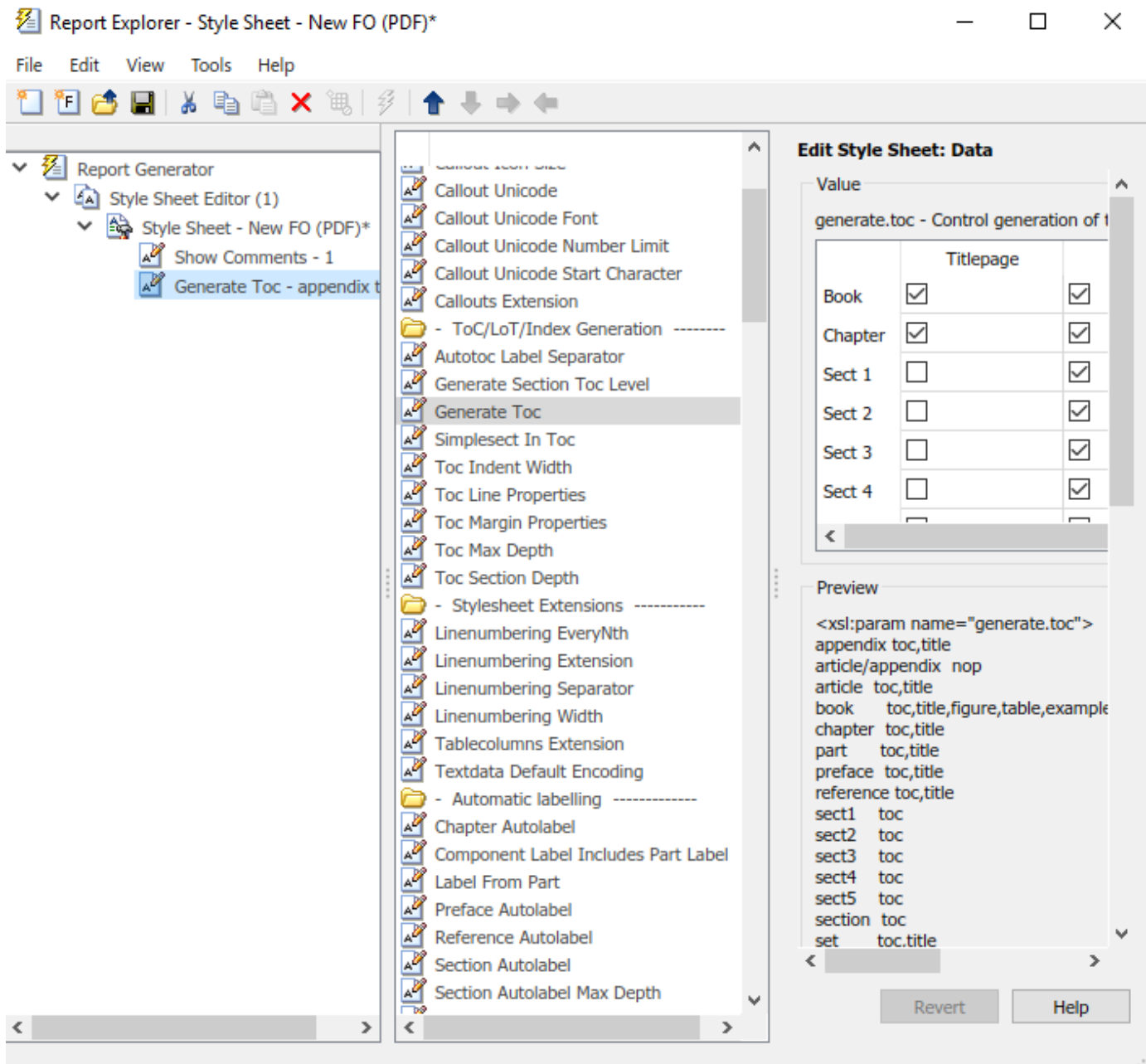
To specify properties for an individual title page element:

- 1 In Properties pane, in the **Include on title page** list, select the title page element.
- 2 Adjust the applicable properties:
  - Layout options — Specify which title page grid row and column in which you which you want the element to appear. To span multiple rows or columns, specify numbers for the **Span row** and **Span column** properties.
  - Format options — For text elements, specify the font size, whether to use bold or italics, text color, and text alignment.
  - Transform options — You can use these options to specify XSLT code to customize the contents and format of title page elements. Use the **XPath** property to specify the path to the XSLT object that you want to modify. Use the **Transform** property to specify the custom content and layout.

### Modify TOC Properties

To change values for generation of the report's table of contents (TOC), select the appropriate values from a matrix of check boxes.

The following figure shows the values for the **Generate Toc** data item on the **PDF** style sheet. Select the check boxes to control the values that appear in the report's title page and table of contents.



### Modify Title Placement Properties

The **Title Placement** data items, which are in the **Miscellaneous** category, control the position of titles for figures and tables.

Selecting one of these data items for editing causes the Properties pane on the right to display possible values in a menu list. Specify whether you want the title to appear before or after a given figure or table.



## Modify Attributes

An attribute is a data item that specifies information for an XML element. An attribute must be a child of an *attribute set*. For more information, see “Edit Attribute Sets” on page 9-17.

---

**Note** The information in the **Help** area of the Properties pane of an attribute describes the set to which the attribute belongs.

---

## Edit Attribute Sets

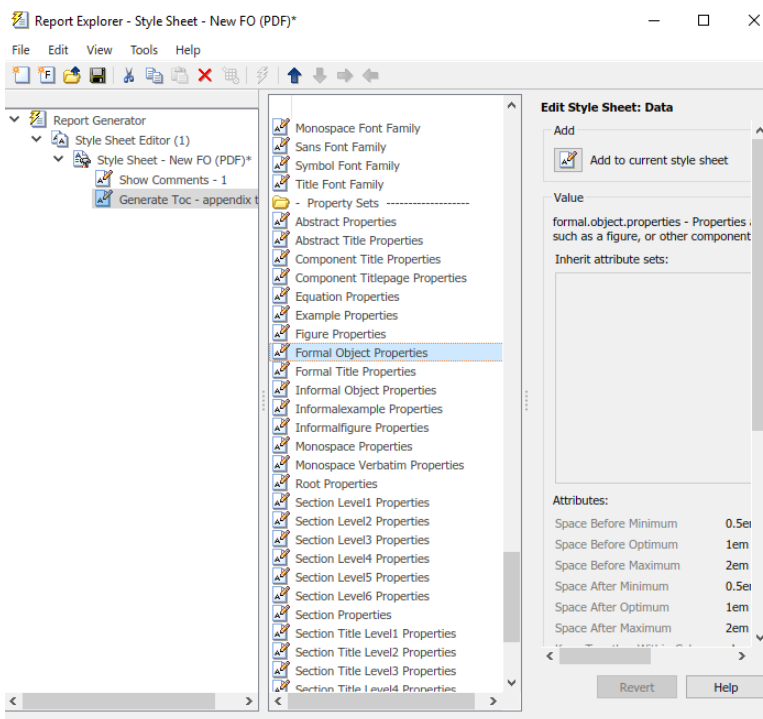
An *attribute set* consists of a group of attributes. Selecting an attribute set in the Outline pane on the left causes the Properties pane to list the attributes that belong to that set.

To edit a specific attribute, expand the attribute set in the Outline pane and select the attribute you want to edit.

To edit the attribute set, type text in the **Inherit attribute sets** area of the Properties pane.

An example of an attribute set is `Formal Object Properties`, a data item in the `Property Sets` category of the default print style sheet for PDF documents.

Here is an example of the Report Explorer showing the `Formal Object Properties` attribute set in the Properties pane.



## Edit Varpair Values

Data items in RTF style sheets appear as `varpair` data items, which are name/value pairs of information. RTF style sheets are the only type of style sheet that includes `varpair` data items.

Edit `varpair` data items as character vectors or as Boolean values. Boolean values appear as `true` (`#t`) and `false` (`#f`).

---

**Note** You cannot edit RTF style sheet data items as XML.

---

**Note** Data of type `varpair` is sometimes represented in style sheets as DSSSL rather than XML. As a result, the code that appears in the **Preview** pane of the Properties pane on the right looks different from code associated with other kinds of MATLAB Report Generator style sheets.

---

### Delete Data Items

To delete a customized data item:

- 1 Right-click the data item in the Outline pane on the left.
- 2 Select **Delete**.

## Style Sheet Cells for Headers and Footers

In this section...
"About Style Sheet Cells and Cell Groups" on page 9-19
"Headers and Footers" on page 9-20
"Add Content to Headers and Footers Using Templates" on page 9-21
"Insert Graphics Files" on page 9-22
"Modify Fonts and Other Properties" on page 9-22

### About Style Sheet Cells and Cell Groups

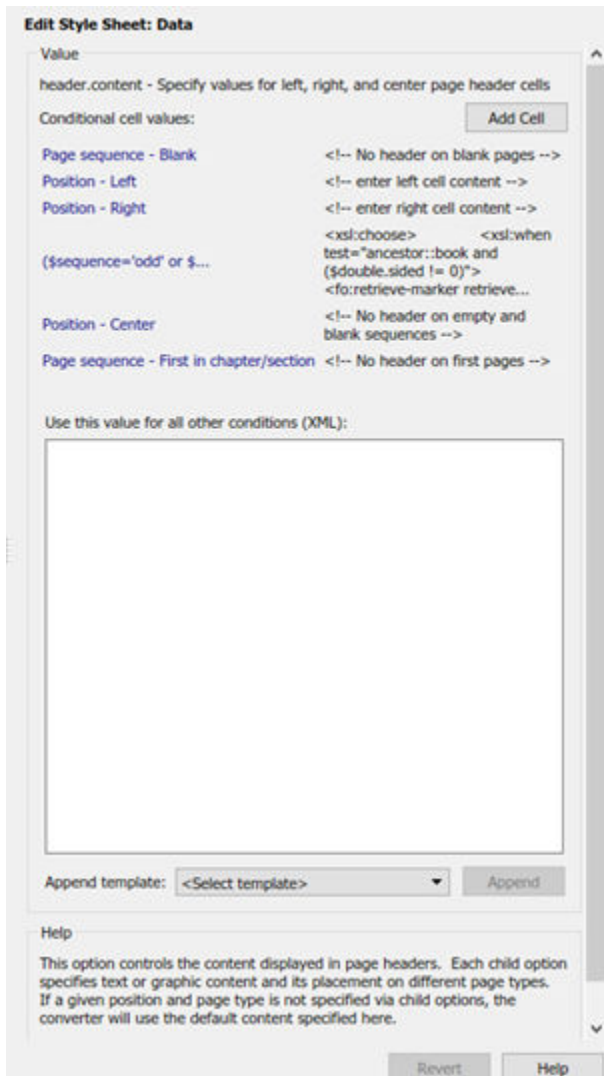
Use *style sheet cells* to specify content of headers and footers in PDF reports.

The MATLAB Report Generator software defines a page as six *cells*. These cells correspond to the left, right, and center of the page's header and the left, right, and center of the page's footer.

A cell group consists of one or more style sheet cells. Two cell groups are available for PDF reports: **Header Content** and **Footer Content**.

The Properties pane for each cell in a cell group lists the group's current style sheet cell definitions. These definitions appear in a two-column list of **Conditional cell values**. The first column displays the name of a *condition*. The second column displays *content* that appears in the report if the specified condition is met.

For example, the style sheet cell `Page sequence - Blank` specifies the content for a blank page; by default, the content is empty. Similarly, `Position - Right` specifies the content for the right side of the header on every page.



You can use many combinations of conditions and values to customize content of headers and footers. The MATLAB Report Generator software provides several predefined conditions that are frequently used. These predefined cells appear in the Properties panes for the Header Content and for Footer Content cell groups.

## Headers and Footers

### Add Content That Satisfies Specified Conditions

You can use the Properties pane of a style sheet cell to specify content that satisfies specified conditions. The Properties pane for a style sheet cell includes the following.

Label	Definition	Description
<b>Condition</b>	Condition that must be met for content to appear in the report	This is a selection list of frequently used and predefined conditions. Select a condition and click <b>Edit</b> to view or change a condition's XML code

Label	Definition	Description
<b>Value (XML)</b>	Content to appear in the report if the condition is met	Modify or create XML code for header or footer content
<b>Append Template</b>	Name of the template that you use to add content	Templates containing XML code that you can use to add content. For more information, see “Add Content to Headers and Footers Using Templates” on page 9-21.

When the File Converter processes a page, it evaluates settings that are relevant to each of the six cells on the page and adds content accordingly. If there are no conditions in effect for a given cell, the File Converter uses the default values for the cell group.

Possible conditions and their values as coded in XML are shown in the following table.

Name of Condition	Possible Values for the Condition	Sample XML Code
\$position	right center left	\$position='right' \$position='center' \$position='left'
\$sequence	odd even first blank	\$sequence=odd \$sequence=even \$sequence=first \$sequence=blank
\$double-sided	0 1	\$double-sided=0 \$double-sided=1
\$pageclass	\$titlepage \$lot \$body	\$pageclass=\$titlepage \$pageclass=\$lot \$pageclass=\$body

Use standard logical operators (such as = , != , and, or) and nested expressions (characters between parentheses are an expression within an expression) to specify *complex conditions*. You can use complex conditions to set the position of headers and footers on pages. You can also use them to specify other settings, such as where in the report the content appears.

## Add Content to Headers and Footers Using Templates

*Templates* are available for adding the following items to headers and footers:

- Text
- Author names
- Page numbers
- Titles for chapters and sections
- Chapter numbering
- Draft information
- Comments
- Graphics

Templates used by the File Converter are Extensible Style Language Transformations (XSLT), which is a language for transforming XML documents into other XML documents. For details about XSLT, see the Web site for the World Wide Web Consortium (W3C®) at <https://www.w3.org/TR/xslt>.

To use a template to specify content for a header or footer:

- 1 In the **Append template** list, select the type of content you want to add.
- 2 Click **Append**.

The Properties pane on the right displays default content for the type you select. Edit the XML code to change the default content.

For example, to specify `text` as the content:

- 1 Select **Text** from the **Append template** list.
- 2 Click **Append**.
- 3 The default value for `xsl:text` is `Confidential`. Edit the value as needed.

## Insert Graphics Files

To add a graphics file to headers or footers in a report, you must:

- 1 Specify the name of the file in the **Header Content** or **Footer Content** style sheet cell.
- 2 Edit the values of the `Region Before Extent` and `Region After Extent` data items. These are located in the **Pagination and General Styles** folder of the **Options** pane for PDF formatting.

For an example of adding a graphic file to a header, see “Add Graphics to Headers in PDF Reports” on page 9-24.

---

**Note** PDF reports only support bitmap (.bmp), jpeg (.jpg), and Scalable Vector Graphics (.svg) images in headers and footers.

---

## Modify Fonts and Other Properties

You cannot use style sheet cells to modify the font family or other such properties of headers and footers. To specify the style of the content in headers and footers, use the **Header Content Properties** and **Footer Content Properties** attribute sets.

Each of these attribute sets is a pagination style data item for PDF style sheets. You can edit a particular attribute in the set by selecting it in the Outline pane on the left.

For an example of modifying font size and other properties of a PDF report, see “Change Font Size, Page Orientation, and Paper Type of a Generated Report” on page 9-27.

## Customized Style Sheets

### In this section...

“Number Pages in a Report” on page 9-23

“Add Graphics to Headers in PDF Reports” on page 9-24

“Change Font Size, Page Orientation, and Paper Type of a Generated Report” on page 9-27

“Edit Font Size as a Derived Value in XML” on page 9-29

### Number Pages in a Report

This example shows how to edit a style sheet cell to number the upper-right side of all pages in the generated report.

- 1 Define a basic style sheet cell in the Header Content cell group with a condition of **right**.
  - a Open a PDF style sheet in the Report Explorer.
  - b Double-click **Header Content** (under **Pagination and General Styles**) in the Options pane in the middle.
  - c Click **Position - Right** in the Properties pane on the right.
- 2 Set the header content to the current page number by selecting **Page number** from the **Append template** selection list.



- 3 Click **Append**.

## Add Graphics to Headers in PDF Reports

This example shows how to include an image in the center of the header of each page in a PDF report, excluding the report's title page and the first page of each chapter. You do this by editing default header content for a PDF style sheet. This example uses the report setup file `mfile-report.rpt`.

You can use any bitmap or jpeg file as image content. You must know the size of the image so that you can allow enough room for it in the header. This example uses the `sample_logo.bmp` image, which is shown here.



---

**Note** PDF reports only support bitmap (.bmp), jpeg (.jpg), and Scalable Vector Graphics (.svg) images.

---

To include this image file in the center of each header in the body of a PDF report:

- 1 Open `mfile-report.rpt` by entering the following at the MATLAB command prompt:  

```
setedit mfile-report
```
- 2 Create a custom style sheet.
  - a Select **Tools > Edit Style Sheet** in the menu bar of the Report Explorer.
  - b Click **New FO (PDF)** in the Properties pane on the right.
  - c As the **Display name**, enter `Logo style sheet for PDF`.
  - d As **Description**, enter `Company logo in center of header`.
  - e Save the style sheet as `logo_style_sheet.rgs` in a folder on your MATLAB path.
- 3 Open the cell group for editing.
  - a Scroll through the Options pane on the left to the **Pagination and General Styles** folder.
  - b Double-click **Header Content** in the Options pane.
  - c Click **Body page - Center** from the list of cells in the Properties pane on the right.

The Properties pane appears as shown.



**Edit Style Sheet: Cell**

Value

If this condition is true, use this value for the current header/footer cell location.

Condition:

Value (XML):

Append template:

Help

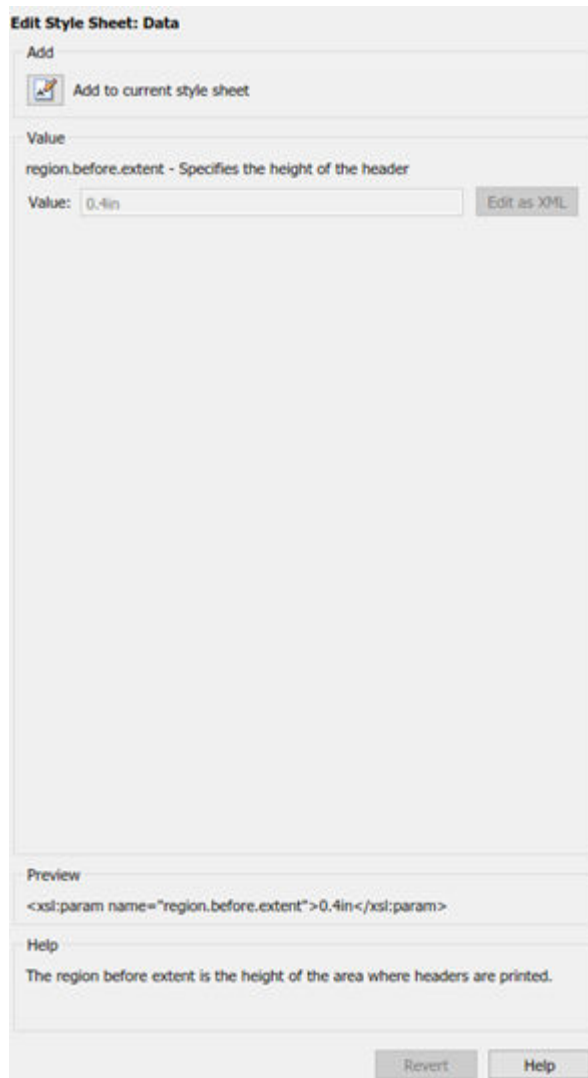
This option specifies page header/footer content and placement. The "Value (XML)" field is XML code which specifies the text or graphics to appear in the header or footer. The "Condition" field controls where and on what type of pages the content is used.

- d Delete the text in the **Value (XML)** field.
- e Select **Graphic** from the **Append template** selection list and click **Append**.

The Properties pane on the right shows the XML code that tells the File Converter to include the graphic.



- 4 By default, the name of the graphic is `logo.bmp`. Change all instances of this name to `sample_logo.bmp` in the **Value (XML)** field.
- 5 Save the style sheet.
- 6 Make sure that the amount of room available in the header is large enough to accommodate the image file.
  - a In the Options pane in the middle, double-click **Region Before Extent**, which is in the **Pagination and General Styles** folder.



- b** By default the value for the height of the header is 0.4 inch. Replace this value with 1.0 in.
  - c** Save the style sheet.
- 7** Generate the report with the new styles.
  - a** Select **mfile-report.rpt** in the Outline pane on the left.
  - b** In the selection lists under the **Report Format and Style Sheet** area of the Properties pane on the right:
    - Specify Acrobat (PDF) for **File format**
    - Specify Logo style sheet for PDF.
  - c** Click **Report** on the toolbar to generate the report.

## Change Font Size, Page Orientation, and Paper Type of a Generated Report

This example shows how to:

- Generate an XML source file without converting it to a supported report format
- Make section headers in a report larger
- Change the report page orientation to landscape
- Change the report paper type to A4

Create a custom style sheet by editing an existing style sheet to change the appearance of the `wsvar-report` report, which is provided with the MATLAB Report Generator software.

- 1 Generate a source file for the report.
  - a Open the report by entering the following command in the MATLAB Command Window:

```
setedit wsvar-report
```
  - b In the **Report Format and Style Sheet** area of the Properties pane, change the format to **DocBook (no transform)**.
  - c Check the **If report already exists, increment to prevent overwriting** check box.
  - d Select **File > Report** to generate the report.

The report-generation process creates an XML source file in the MATLAB Editor.

- 2 Convert the report to PDF format.
  - a Select **Tools > Convert Source File** from the Report Explorer menu bar to open the File Converter.
  - b From the **Source file** selection list, enter **wsvar-report0.xml**.
  - c From the **File format** selection list, select **Acrobat (PDF)**.
  - d From the **Style Sheet** selection list, select **Unnumbered Chapters and Sections**.
  - e Click **Convert File**.

The MATLAB Report Generator software converts the XML source file for `wsvar-report` to PDF format, and then opens the PDF document.

- 3 Make the report headers more prominent.
  - a In the File Converter, click **Edit**.

The Report Explorer displays the **Unnumbered Chapters and Sections** style sheet.
  - b In the Properties pane on the right, enter **Custom Large Section Headers** as the style sheet name.
  - c Enter the description **No chapter and section numbering, larger section titles**.
  - d In the Outline pane on the left, select the **Custom Large Section Headers** style sheet.
  - e In the Options pane in the middle, select **Section Title Level 1 Properties**.
  - f In the Properties pane on the right, click **Add to current style sheet**.

The **Section Title Level 1 Properties** data item appears in the Outline pane on the left as a child of the **Custom Large Section Headers** style sheet.

- g In the Properties pane on the right, select the **Font Size** attribute.

The Properties pane on the right displays an XML expression specifying font size as a multiple of the **Body Font Size** attribute.

- h** Click **Edit as string**.

The MATLAB Report Generator software converts the XML expression to text, which appears in a pane labeled **Value**.

- i** Enter the value 18pt.

The size of the font is now fixed at 18 points, rather than being a multiple of the body font size attribute.

- j** Select **File > Save** to save the style sheet.

- k** Save the style sheet as `customheader.rgs`, in a folder in your MATLAB path.

The `customheader.rgs` style sheet appears as an available style sheet in the Options pane in the middle of the Report Explorer. It also appears as an option in the File Converter.

- 4** Use the new style sheet to convert the current XML source file.

- a** In the **Edit Style Sheet: Main** Properties pane on the right, click **Send to File Converter**

The File Converter appears, with the `customheader.rgs` style sheet selected.

- b** Click **Convert file**.

- 5** Change page orientation and paper type.

- a** On the File Converter Properties pane, click **Edit**.

- b** In the Options pane on the left, double-click the **Page Orientation** data item.

- c** In the Properties pane on the right, use the selection list to change the value of the data item to Landscape.

- d** In the Options pane in the middle, double-click **Paper Type** in the **Pagination and General Styles** folder.

- e** In the Properties pane on the right, select **A4** from the selection list.

- f** Save the style sheet.

- 6** Generate the report `wsvar-report.xml` in PDF format using `customheader.rgs`.

The PDF report appears with horizontally oriented pages of slightly different dimensions.

## Edit Font Size as a Derived Value in XML

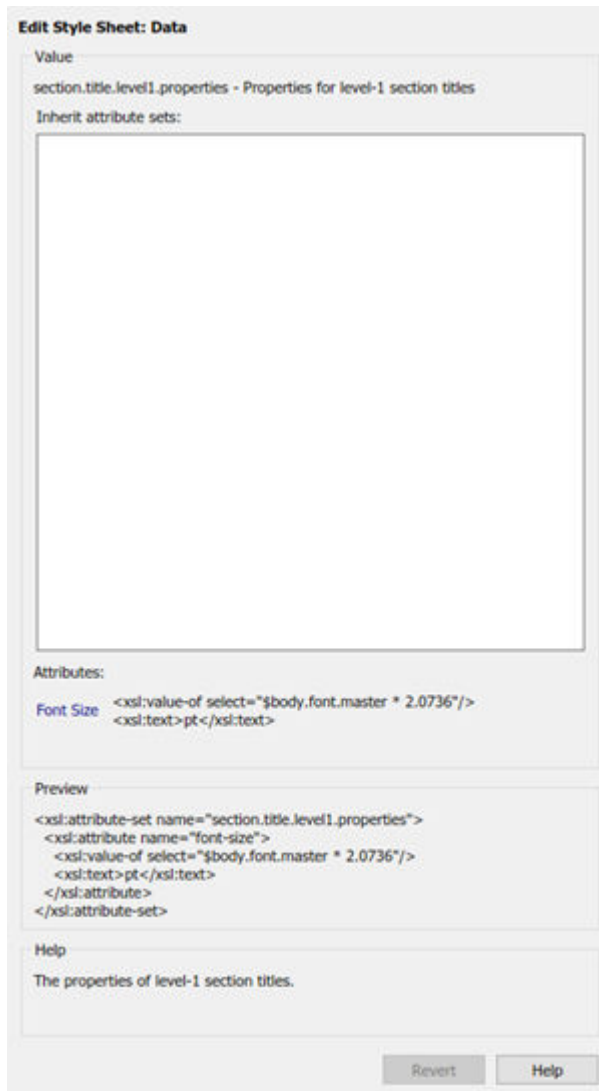
This example shows how to change the font size in a report to a value derived from other values. You do this by editing the PDF report's XML source directly.

- 1** Open the default print style sheet for PDF documents.

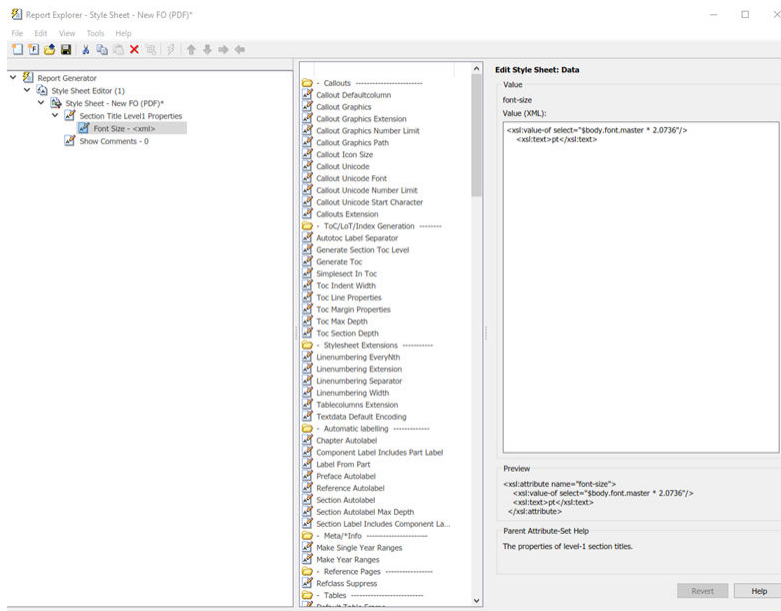
- 2** In the Options pane in the middle, select and expand the **Property Sets** folder.

- 3** In the Options pane, double-click the **Section Title Level1 Properties** data item.

The Properties pane on the right appears as follows.



- 4 In the **Attributes** area of the Properties pane on the right, click **Font Size - <xml>**.  
The Report Explorer looks as follows.



The **font size** value is a product of `$body.font.master` and `2.0736`. To change the font size to a larger size, change the multiplication factor to `3.0736`.

---

**Tip** You specify the value for the `$body.font.master` data item in the **Body Font Master** property. This property is in the **Pagination and General Styles** category in the Options pane in the middle. The default value of this data item is 10. Changing this value causes the derived values to change accordingly.

---

## Configure PDF Fonts

### In this section...

“PDF Font Support for Languages” on page 9-32  
 “Identifying When to Specify a Font” on page 9-32  
 “Style Sheets Override PDF Font Mapping” on page 9-33  
 “Non-English PDF Font Mapping Tasks” on page 9-33  
 “lang\_font\_map.xml File” on page 9-33  
 “Locate Non-English Fonts” on page 9-35  
 “Add or Modify Language Font Mappings” on page 9-36  
 “Specify the Location of Font Files” on page 9-36

### PDF Font Support for Languages

For PDF output, MATLAB Report Generator comes configured with default fonts: a serif, a sans serif, and dingbats. It also comes configured to use the font for your language, based on your locale. You can map to a different font for your locale.

When generating PDF reports, the MATLAB Report Generator uses a font capable of rendering text in these languages:

- English
- Japanese
- Korean
- Russian (Cyrillic)

The Report Generator uses a font map to determine the font appropriate for a particular locale. The font map specifies a default set of fonts. You can modify the map to:

- Change the fonts used for a particular locale
- Add support for locales other than the default locales

The language font map specifies the font to use on a specific platform (for example, Windows) and locale for basic report elements such as body text.

### Identifying When to Specify a Font

If a required non-English font is missing for a report, the generated text includes pound sign characters (#). For example:

```
#####
```



## Style Sheets Override PDF Font Mapping

PDF style sheets for the MATLAB Report Generator specify fonts for body text, copyright, quotes, symbols, dingbats, monospace, sans serif, and titles.

The PDF style sheet settings override the PDF font mapping entries.

If you do not specify a PDF style sheet, then you can use PDF language font mapping entries to change the default fonts for English reports.

## Non-English PDF Font Mapping Tasks

To add or modify non-English PDF font mapping specifications:

- “Locate Non-English Fonts” on page 9-35
- “Add or Modify Language Font Mappings” on page 9-36
- “Specify the Location of Font Files” on page 9-36

## lang\_font\_map.xml File

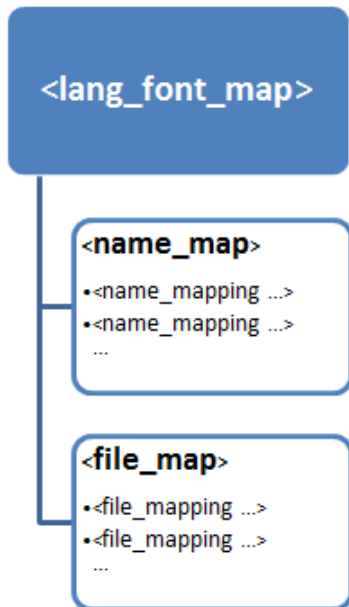
Use an XML editor with the `lang_font_map.xml` file to enter all the PDF font mappings for your reports.

Installing the MATLAB Report Generator software loads the `lang_font_map.xml` file in the following location:

```
<matlabroot>/toolbox/shared/rptgen/resources/fontmap
```

The `lang_font_map.xml` file includes two sections:

- `name_map` — Contains `name_mapping` elements that specify the name of the font, the language, and the font usage in the report (for example, body text).
- `file_map`— Contains entries for the location of the font files for the fonts specified in the `name_map`.



For example, the following `lang_font_map.xml` file includes `name_map` and `file_map` entries that provide basic PDF font support for Japanese (ja), Korean (ko), and Russian (ru).

### lang\_font\_map.xml example

```

<?xml version="1.0" encoding="UTF-8" ?>
<lang_font_map>
  <name_map>
    <name_mapping lang="ja" platform="win" usage="body">MS Gothic</name_mapping>
    <name_mapping lang="ja" platform="win" usage="monospace">MS Gothic</name_mapping>
    <name_mapping lang="ja" platform="win" usage="sans">MS Gothic</name_mapping>
    <name_mapping lang="ja" platform="win" usage="title">MS Gothic</name_mapping>

    <name_mapping lang="ko" platform="win" usage="body">Gulim</name_mapping>
    <name_mapping lang="ko" platform="win" usage="monospace">Gulim</name_mapping>
    <name_mapping lang="ko" platform="win" usage="sans">Gulim</name_mapping>
    <name_mapping lang="ko" platform="win" usage="title">Gulim</name_mapping>

    <name_mapping lang="ru" platform="win" usage="body">Arial Unicode MS</name_mapping>
    <name_mapping lang="ru" platform="win" usage="monospace">Arial Unicode MS</name_mapping>
    <name_mapping lang="ru" platform="win" usage="sans">Arial Unicode MS</name_mapping>
    <name_mapping lang="ru" platform="win" usage="title">Arial Unicode MS</name_mapping>

    <name_mapping lang="en" platform="glx" usage="body">FreeSerif, Regular</name_mapping>
    <name_mapping lang="en" platform="glx" usage="monospace">FreeMono, Regular</name_mapping>
    <name_mapping lang="en" platform="glx" usage="sans">FreeSans, Regular</name_mapping>
    <name_mapping lang="en" platform="glx" usage="title">FreeSerif, Bold</name_mapping>

    <name_mapping lang="ru" platform="mac" usage="body">Arial Unicode MS</name_mapping>
    <name_mapping lang="ru" platform="mac" usage="monospace">Arial Unicode MS</name_mapping>
    <name_mapping lang="ru" platform="mac" usage="sans">Arial Unicode MS</name_mapping>
    <name_mapping lang="ru" platform="mac" usage="title">Arial Unicode MS</name_mapping>
  </name_map>

  <file_map>
    <file_mapping lang="ja" platform="win" name="MS Gothic">msgothic.ttc</file_mapping>
    <file_mapping lang="ja" platform="win" name="MS PGothic">msgothic.ttc</file_mapping>

    <file_mapping lang="ko" platform="win" name="Gulim">gulim.ttc</file_mapping>

    <file_mapping lang="en" platform="glx" name="FreeSerif, Regular">FreeSerif.ttf</file_mapping>
    <file_mapping lang="en" platform="glx" name="FreeMono, Regular">FreeMono.ttf</file_mapping>
    <file_mapping lang="en" platform="glx" name="FreeSans, Regular">FreeSans.ttf</file_mapping>
    <file_mapping lang="en" platform="glx" name="FreeSerif, Bold">FreeSerifBold.ttf</file_mapping>

    <file_mapping lang="ru" platform="mac" name="Arial Unicode MS">Arial Unicode.ttf</file_mapping>

```

```
</file_map>
</lang_font_map>
```

## Locate Non-English Fonts

The system from which you generate a report using the language font map must have access to the appropriate non-English fonts.

Use one of these font formats for non-English font support:

- Type 1 (PostScript®)
- TrueType
- OpenType®

Fonts in other formats, such as bitmap fonts for the X Window System (X11), produce poor MATLAB Report Generator report output.

Some TrueType fonts are grouped into packages called TrueType Collections. To specify a collection in the language font map file, specify the individual font within the TrueType Collection.

In addition to the font name, the weight (e.g., bold) and slant (e.g., italic, oblique) may distinguish one font from another in the same family.

The approach you use to identify font names depends on your computer platform.

### Font names on Windows

To identify a TrueType font name on Windows systems:

- 1 Navigate to the font folder (usually C:\Windows\Fonts).
- 2 If the font is a simple TrueType (not a collection), in the window, right-click the font and choose **Properties** to see the name of the file containing that font.
- 3 If the font is a TrueType Collection, right-click to open the collection, optionally in a new window. Each constituent font appears, with its name. Use the name of the constituent font, not the name of the whole collection.
  - a Right-click any of constituent font and select **Properties**. The properties box displays the name of the file containing that font.

### Font names on Mac OS X

Mac OS X provides an application called Font Book (in the /Applications folder) that provides information about available fonts on the system. The application shows all the fonts on your system. Hover over a specific font to see a datatip with the font name and the path to the font.

### Font names on Linux

Linux distributions use a variety of conventions for the location of fonts, or how those font folders can be found. By default, MATLAB Report Generator searches these folders, in this order:

- 1 /.fonts/
- 2 /usr/local/share/fonts/
- 3 /usr/X11R6/lib/fonts/

#### **4** /usr/share/fonts/

You can specify alternative folders in the `fonts.conf` file (in the `/etc/fonts/` folder).

## **Add or Modify Language Font Mappings**

In the `name_map` section of the `lang_font_map.xml` file, add a separate `name_mapping` entry for each combination of language, font, and usage that you want in PDF reports.

Each `name_mapping` element has three attributes:

- `lang` specifies the two letter ISO 639-1 code corresponding to the language of the report.
- `platform` specifies the operating system platform:
  - `win` — Windows
  - `mac` — Mac OS X
  - `glnx`— Linux
- `usage` specifies the kind of report element or font:
  - `body`
  - `title`
  - `monospaced`
  - `sans` (sanserif)

The text of the `name_mapping` element is a font name, as specified in an XSL-FO style sheet.

Here is an example `name_mapping` entry:

```
<name_mapping_lang="ja" platform="win" usage="body">MS Gothic</name_mapping>
```

## **Specify the Location of Font Files**

In the `file_map` section, add a `file_mapping` entry that identifies the location of the font file for each font that you include in the `name_map` section.

Each of the platforms (Windows, Mac, and Linux) has a different default search path for fonts. If the `lang_font_map.xml` file does not contain a full file path for a font, the MATLAB Report Generator uses a platform-specific approach to search for the font.

### **Windows Font File Locations**

On Windows platforms, the MATLAB Report Generator searches for fonts in `<windir>/Fonts`, where `windir` is an operating system environment variable. The typical location is `C:\Windows` or `C:\Winnt`.

### **Mac Font File Locations**

On Mac OS X platforms, fonts are generally in one of these folders:

- `~/Library/Fonts`
- `/Library/Fonts`

- Network/Library/Fonts
- System/Library/Fonts
- System/Folder/Fonts

### **Linux Font File Locations**

On Linux platforms, the convention for locating fonts can differ, depending on the Linux distribution. The MATLAB Report Generator follows the Debian® convention of finding the list of font folders in the `/etc/fonts/fonts.conf` file.

If the MATLAB Report Generator does not find the `fonts.conf` file in `/etc/fonts/` folder, it searches the following folders, in the following order:

- 1** `/.fonts`
- 2** `/usr/local/share/fonts`
- 3** `/usr/X11R6/lib/fonts`
- 4** `/usr/share/fonts`

Because of the variety of conventions used in different Linux distributions, consider using full file paths in `file_mapping` elements.



# Components

---

## Array-Based Table

Convert rectangular array into table and insert it into report

### Description

This component converts a rectangular cell array into a table and inserts the table into the report.

### Table Content

- **Workspace variable name:** Specifies the workspace variable name with which to construct the table.
- **Collapse large cells to a single description:** Consolidates large cells into one description.

### Formatting Options

- **Table title:** Specifies the title of your table.
- **Cell alignment:**
  - left
  - center
  - right
  - double justified
- **Column widths:** Inputs a vector with  $m$  elements, where  $m$  equals the number of columns in the table. Column sizing is relative and normalized to page width. For example, say that you have a 2-by-3 cell array and input the following into the **Column widths** field:

```
[1 2 3]
```

The report output format for the cell array is such that the second column is twice the width of the first column, and the third column is three times the width of the first column. If the vector is greater than the number of columns in the table, the vector is truncated so that the number of elements equals the number of columns. If  $m$  is less than the number of columns in the table, the vector is padded with 1s so that the number of elements equals the number of columns.

If you use this field, it is recommended that you specify a width for each column. Any width not specified defaults to 1. MATLAB displays a warning when defaulting any unspecified column width to 1.

- **Table grid lines:** Displays grid lines, which create borders between fields, in the table.
- **Table spans page width (HTML only):** Sets the table width to the width of the page on which it appears.

### Header/Footer Options

Designating a row as a header or footer row causes the contents of the row to appear in boldface.

- **Number of header rows:** Specifies the number of header rows.



- **Footer list:**
  - `No footer`: Specifies no footers for the report.
  - `Last N rows are footer`: Enables you to select a footer that is different from your header.

### Example

Consider the following cell array in the MATLAB workspace:

```
{'foo', 'bar'; [3], [5]}
```

Its cell table in the report appears as follows.

foo	bar
3	5

Note that the table has no headers or footers and no title.

## Insert Anything into Report?

Yes. Table.

### Class

`rptgen.cfr_table`

### See Also

[Table](#), [Table Body](#), [Table Column Specification](#), [Table Entry](#), [Table Footer](#), [Table Header](#), [Table Row](#), [Chapter/Subsection](#), [Empty Component](#), [Image](#), [Link](#), [List](#), [Paragraph](#), [Text](#), [Title Page](#)

## Axes Loop

Run child components for all axes objects in MATLAB workspace

### Description

The Axes Loop component runs its child components for all axes objects in the MATLAB workspace. For information about working with looping components, see “Logical and Looping Components” on page 6-17.

### Object Selection

- **Loop type:**
  - All axes: Loops on all axes objects.
  - Current axes: Loops on the currently selected axes object.
- **Exclude objects which subclass axes:** Excludes objects, such as legends and color bars, from the loop.
- **Loop Menu:**
  - Loop on axes with handle visibility "on": Loops only on visible axes objects.
  - Loop on all axes: Loops on all axes objects.
- **Search terms:** Specifies search terms for the loop. For example, to search for Tag and My Data, enter "Tag" , "My Data".

### Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each object found in the loop.
- **Display the object type in the section title:** Automatically inserts the object type into the section title in the generated report.
- **Create link anchor for each object in loop:** Create a link target on each object in the loop so that other parts of the report can link to it.

### Insert Anything into Report?

Yes, inserts a section if you select the **Create section for each object in loop** option.

### Class

rptgen\_hg.chg\_ax\_loop

**See Also**

Axes Snapshot, Figure Loop, Figure Snapshot, Graphics Object Loop, Handle Graphics Linking Anchor, Handle Graphics Name, Handle Graphics Parameter, Handle Graphics Property Table, Handle Graphics Summary Table

## Axes Snapshot

Insert image of selected MATLAB axes objects into the generated report

### Description

Inserts an image of selected MATLAB axes objects into the generated report.

### Format

- **Image file format:** Specifies the image file format. Select Automatic HG Format to automatically choose the format best suited for the specified report output format. Otherwise, choose an image format that your output viewer can read. Automatic HG Format is the default option. Options include:
  - Automatic HG Format (uses the Handle Graphics file format selected in the Preferences dialog box)
  - Bitmap (16m-color)
  - Bitmap (256-color)
  - Black and white encapsulated PostScript
  - Black and white encapsulated PostScript (TIFF)
  - Black and white encapsulated PostScript2
  - Black and white encapsulated PostScript2 (TIFF)
  - Black and white PostScript
  - Black and white PostScript2
  - Color encapsulated PostScript
  - Color encapsulated PostScript (TIFF)
  - Color encapsulated PostScript2
  - Color encapsulated PostScript2 (TIFF)
  - Color PostScript
  - Color PostScript2
  - JPEG high quality image
  - JPEG medium quality image
  - JPEG low quality image
  - PDF image
  - PNG00000000000000000000 24-bit image
  - TIFF - compressed
  - TIFF - uncompressed
  - Windows metafile
- **Capture figure from screen:** Captures the figure for the generated report directly from the screen. Options include:

- **Client area only:** Captures part of the figure.
- **Entire figure window:** Captures the entire figure window.

## Print Options

- **Paper orientation:**

- Landscape
- Portrait
- Rotated
- **Use figure orientation:** Uses the orientation for the figure, which you set with the `orient` command.
- **Full page image (PDF only):** In PDF reports, scales images to fit the full page, minimizes page margins, and maximizes the size of the image by using either a portrait or landscape orientation.

For more information about paper orientation, see the `orient` command in the MATLAB documentation.

- **Image size:**

- **Use figure PaperPositionMode setting:** Sets the image size in the report to the `PaperPositionMode` property of the figure. For more information about paper position mode, see `orient`.
- **Automatic (same size as onscreen):** Sets the image in the report to the same size as it appears on the screen.
- **Custom:** Specifies a custom image size. Specify the image size in the **Size** field and **Units** list.
- **Size:** Specifies the size of the figure snapshot in the format `[w h]` (width, height). This field is active only if you choose **Custom** in the **Image size** selection list.
- **Units:** Specifies the units for the size of the figure snapshot. This field is active only if you choose `Set image size` in the **Custom** selection list.
- **Invert hardcopy:** Sets the `InvertHardcopy` property of Handle Graphics figures. This property inverts colors for printing; that is, it changes dark colors to light colors and vice versa. Options include:
  - **Automatic:** Automatically changes dark axes colors to light axes colors. If the axes color is a light color, it is not inverted.
  - **Invert:** Changes dark axes colors to light axes colors and vice versa.
  - **Don't invert:** Does not change the colors in the image displayed on the screen for printing.
  - **Use figure's InvertHardcopy setting:** Uses the `InvertHardcopy` property set in the Handle Graphics image.
  - **Make figure background transparent:** Makes the image background transparent.

## Display Options

- **Scaling:** Controls size of the image, as displayed in a browser. Making an image larger using this option does not affect the storage size of the image, but the quality of the displayed image may decrease as you increase or decrease the size of the displayed image.

Generally, to achieve the best and most predictable display results, use the default setting of `Use image size`.

- `Use image size`: Causes the image to appear the same size in the report as on screen (default).
- `Fixed size`: Specifies the number and type of units.
- `Zoom`: Specifies the percentage, maximum size, and units of measure.
- **Size**: Specifies the size of the snapshot in the format `w h` (width, height). This field is active only if you choose `Fixed size` in the **Scaling** list.
- **Max size**: Specifies the maximum size of the snapshot in the format `w h` (width, height). This field is active only if you choose `Zoom` from the **Scaling** list.
- **Units**: Specifies the units for the size of the snapshot. This field is active only if you choose `Zoom` or `Fixed size` in the **Image size** selection list.
- **Alignment**: Only reports in PDF or RTF format support this property. Options are:
  - Auto
  - Right
  - Left
  - Center
- **Title**: Specifies text to appear above the snapshot.
- **Caption**: Specifies text to appear under the snapshot.

## Insert Anything into Report?

Yes. Image.

## Class

`rptgen_hg.chg_ax_snap`

## See Also

Axes Loop, Figure Loop, Figure Snapshot, Graphics Object Loop, Handle Graphics Linking Anchor, Handle Graphics Name, Handle Graphics Parameter, Handle Graphics Property Table, Handle Graphics Summary Table

## Chapter/Subsection

Group portions of report into sections with titles

### Description

This component groups portions of the report into sections. Each section has a title and content.

The following rules apply to this component:

- Child components appear inside the section created by this component.
- Selecting the **Get title from first child component** check box prevents this component from accepting paragraph-level children. In this case, this component's first child must be a **Text** component.
- This component can have **Chapter/Subsection** components as its children.
- Sections can be nested. There are seven levels of nesting possible. The seventh nested section in the report is untitled, although the child components of this section include information into the report.

### Chapter Numbering

Chapter and subsection numbering depends on the template you select for template-based output, such as Word (from `template`), or the style sheet you select for style-sheet-based output, such as Adobe Acrobat.

To number chapters only in template-based output, select `Default Word Template`, `Default HTML Template`, `Default Single-File HTML Template`, or `Default PDF Template`, depending on the output type. To number both chapters and subsections, select the `Default Numbered` template. You can customize these templates to generate other numbering schemes. For example, to generate unnumbered chapters, delete the `rgChapterTitleNumber` hole in the `rgChapterTitle` library template in your report's main template.

To generate unnumbered chapters and sections in style-sheet-based output, select `Unnumbered Chapters and Sections` as your report's style sheet. To number chapters and sections, select `Numbered Chapters and Sections` as your report's style sheet. You can generate other numbering schemes by customizing these style sheets.

### Customize Word Chapter or Section Title Styles

You can create custom styles in your Word template to change the appearance of chapter or section titles.

- 1 In your Word template, define styles for the title prefix (for example, the word `Chapter` or `Section`), the title number, and the title text. Name them in the form `MyRootPrefix`, `MyRootNumber`, and `MyRootText`, for example:

```
myChapterTitlePrefix
```

```
myChapterTitleNumber
```

```
myChapterTitleText
```

- 2 In the Chapter/Subsection dialog box in Report Explorer, set **Style Name** to your value for MyRoot, for example myChapterTitle.

When you generate your report, the MATLAB Report Generator uses your styles instead of the default styles.

## Section Title

- **Title:** Specifies a title to display in the generated report:
  - **Automatic:** Automatically generates a title.
  - **Custom:** Specifies a custom title.
- **Style Name:** Specifies the style to use with the chapter or section title. To specify a style:
  - 1 Set the report's **File format** to one of the from `template` options, for example, `Direct PDF (from template)`.
  - 2 In the Chapter/Subsection properties dialog box, set **Title** to `Custom`.
  - 3 Set **Style Name** to `Custom`.
  - 4 In the **Style Name** text box, type a style name.

To take effect, the style you specify must be defined in the template that you use to generate the report. For Word documents, do not use the built-in styles. If you want to use a built-in style, create your own style based on the built-in style, for example MyHeading1.

For more information about template styles, see “Report Templates” on page 7-2.

- **Numbering:** Specifies a numbering style for the report:
  - **Automatic:** Numbers by context.
  - **Custom:** Allows you to create your own numbering style.
- **Section Type:** Shows you in which level a selected section resides.

## Insert Anything into Report?

Yes. Chapter or section.

## Class

`rptgen.cfr_section`

## See Also

[Empty Component](#) | [Image](#) | [Link](#) | [List](#) | [Paragraph](#) | [Table](#) | [Text](#) | [Title Page](#)

## Topics

“Customize Microsoft Word Component Templates” on page 7-18



# Code

Insert text formatted as code in a paragraph

## Description

This component inserts text formatted as code in a paragraph. The purpose of the component is to distinguish code, such as variable and property names, from other text in a paragraph. Its default style, `rgCode`, uses a monospace font instead of the default proportional font used by a **Paragraph** or **Text** component. The **Code** component also preserves white space (line feeds and spaces) in HTML and Word output. You can use this component to insert code in paragraphs in HTML, Word, and PDF reports. You can also use it to insert code in table entries and at the report body level in HTML reports, for example, `web` (HTML) reports.

---

**Note** This component applies only to template-based reports, such as `Direct PDF` (from `Template`). In XSL style sheet-based reports, for example `web` (HTML), it serves as a `Text` component.

---

## Properties

- **Code to include in report:** Specifies the code to include in the paragraph.
- **Style Name:** Specifies the name of a style used to format the code. By default, the style name is `rgCode`, which is defined in the Report Explorer default document conversion templates. To override the default style:
  - 1 Set **Style Name** to `Specify`.
  - 2 In the **Style Name** text box, type a style name.

To use the custom style you specify, it must be defined as a span style in the HTML or PDF template used with this report. For a Word template, it must be defined as a character or linked paragraph/character style. For more information about template styles, see “Report Templates” on page 7-2.

## Style

- **Bold:** Makes the text bold.
- **Italic:** Makes the text italic.
- **Underline:** Underlines the text.
- **Color:** Specifies the color of the text.
  - Select a color from the list of colors.
  - Enter a hexadecimal RGB value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.
  - Enter `%<expr>`, where `expr` is a MATLAB expression that evaluates to a color name or a hexadecimal RGB value.

---

**Note** These format options override any corresponding formats defined in the custom style you specify. For example, selecting **Bold** makes the text bold, even if the specified style uses regular weight text.

---

## **Insert Anything into Report?**

Yes. Inline text formatted as code.

For example, if you include a Code component with "Code      component    text" in a paragraph, the output appears exactly as you have typed it, preserving all of the spaces.

### **Class**

```
rptgen.cfr_code
```

### **See Also**

Preformatted, Text, Paragraph

# Comment

Insert comment into XML source file created by report generation process

## Description

This component inserts a comment into the XML source file created by the report-generation process. This comment is not visible in the generated report.

This component can have children. Child components insert their output into the XML source file, but this does not appear in the generated report.

To make comment text appear in the report:

- 1 Edit the XML source file (which has the same name as your report file, but has a `xml` extension).
- 2 Find the `comment` area in the XML source file by locating the comment tags `<--` and `-->`.
- 3 Remove the comment tags.
- 4 Convert the XML source file using the `rptconvert` command.

## Properties

- **Comment text:** Specifies comments to include in the report.
- **Show comment in Generation Status window:** Displays comments in the **Generation Status** tab while the report generates.
- **Status message priority level:** Specifies the priority level of the status messages that appear during report generation. Priority options range from **1 Error messages only** to **6 All messages**. The default is **3 Important messages**. This option is only available if you select the **Show comment in Generation Status window** option.

## Insert Anything into Report?

No. This component inserts comments, which can appear in the report, into the report's XML source file.

## Class

`rptgen.crg_comment`

## See Also

"Convert XML Documents to Different File Formats" on page 5-13, **Import File**, **Nest Setup File**, **Stop Report Generation**, **Time/Date Stamp**

## DOCX Page Layout

Page layout in a Word report

### Description

This component generates a page layout definition for a section of a Microsoft Word report. The page layout definition specifies the size and orientation of pages in the section, the sizes of the section's page margins, and the format and starting value of the section's page numbers.

You can create instances of this component interactively or from report templates. For more information, see "Define Page Layouts in a Form-Based Report Setup" on page 16-4.

### Page Numbering

- **First page number:** Number of the first page in this page layout section.
  - **Auto:** The number of the first page of this layout. If you are using a template, the value is the first page number defined in the template from which this layout was generated. If you create this layout interactively, numbering continues from the previous page layout section.
  - **Specify:** Specify the first page number as an integer.
- **Page number format:** Format of the page numbers in this page layout section:
  - **None:** Use Arabic numerals, the default formatting for page numbers.
  - **Lowercase alphabetic**
  - **Uppercase alphabetic**
  - **Lowercase Roman numerals**
  - **Uppercase Roman numerals**
  - **Arabic numerals**
  - **Number with dashes**
  - **Hebrew numerals**
  - **Hebrew alphabetic**
  - **Arabic alphabetic**
  - **Arabic abjad numerals**
  - **Thai letters**
  - **Thai numerals**
  - **Thai counting system**
- **Section break:** Where to start this section:
  - **Same Page:** Start this page layout on the same page as the last page of the previous section.
  - **Next Page:** Start this page layout on a new page immediately following the last page of the previous section.
  - **Odd Page:** Start this page layout on a new page, immediately after the last page of the previous section. If the previous section ended on an odd page, insert an empty page at the end of the last section.

- **Even Page:** Start this page layout on a new page, immediately after the last page of the previous section. If the previous section ended on an even page, insert an empty page at the end of the previous section.

## Page Margin Options

- **Page Margin:**
  - **Auto:** If this layout component was generated from a template, use the margin values specified by the template. Otherwise, use default values.
  - **Specify:** Specify the size of the page margins in the form `valueUnits`. Use any of these values for units:
    - `cm` — centimeters
    - `in` — inches
    - `mm` — millimeters
    - `pc` — picas
    - `px` — pixels (default)
    - `pt` — points
  - **Top:** Size of top page margin. The default value is `1in`.
  - **Bottom:** Size of bottom page margin. The default value is `1in`.
  - **Left:** Size of left page margin. The default value is `1in`.
  - **Right:** Size of right page margin. The default value is `1in`.
  - **Header:** Size of header area. The default value is `0.5in`.
  - **Footer:** Size of footer area. The default value is `0.5in`.
  - **Gutter:** Size of gutter (area for binding pages). The default value is `0px`.

## Page Size Options

- **Page Size:** If this layout component was generated from a template, selecting `Auto` uses the values specified in the template. Otherwise, selecting `Auto` uses default values.

Select `Specify` to enter your own page height, width, or orientation. Specify the units in the form `valueUnits`. Use any of these values for units:

- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `px` — pixels (default)
- `pt` — points

## Class

`rptgen.cform_docx_page_layout`

**See Also**

Page Footer, Page Header, PDF Page Layout

# Empty Component

Group components to move, activate, or deactivate them, or create blank space in list

## Description

This component does not insert anything into the generated report. It can have any component as a child. You can use it to group components together so that you can easily move, activate, or deactivate them, or create a blank space in a list.

If the MATLAB Report Generator software does not recognize a given component when loading a report setup file, it replaces the unrecognized component with this component.

## Insert Anything into Report?

No.

## Class

`rptgen.crg_empty`

## See Also

Chapter/Subsection, Image, Link, List, Paragraph, Table, Text, Title Page

## Evaluate MATLAB Expression

Evaluate specified MATLAB expression

### Description

This component evaluates a specified MATLAB expression. You can include code and/or command-line output in the report.

### Properties

- **Insert MATLAB expression in report:** Causes the MATLAB expression that this component evaluates to appear in the report.
- **Display command window output in report:** Includes the command window output that results from the evaluation of the specified MATLAB expression.
- **Expression to evaluate in the base workspace:** Specifies the expression to evaluate in the MATLAB workspace. .

If you are using Simulink Report Generator, then you can use functions such as `Rptgen.getReportedBlock` to filter the modeling elements on which to report and to perform special reporting on specific elements. For more information, in the Simulink Report Generator documentation, see “Loop Context Functions” (Simulink Report Generator).

- **Evaluate this expression if there is an error:** Evaluates another MATLAB expression if the specified expression produces an error. You must enter in this field the expression to evaluate in case of an error.

If you do not change the default error handling code, then when you generate the report, and there is an error in the MATLAB code that you added:

- If you clear **Evaluate this expression if there is an error** check box, then the complete report is generated, without displaying an error message at the MATLAB command line.
- If you select **Evaluate this expression if there is an error** check box, then the complete report is generated and an error message appears at the MATLAB command line.

To stop report generation when an error occurs in the MATLAB code that you added, change the second and third lines of the following default error handling code, as described below:

```
warningMessageLevel = 2;  
displayWarningMessage = true;  
failGenerationWithException = false;  
failGenerationWithoutException = false;
```

To stop report generation and display an exception, change the default code to:

```
displayWarningMessage = false;  
failGenerationWithException = true;
```

To stop report generation without displaying an exception, change the default code to:

```
displayWarningMessage = false;  
failGenerationWithoutException = true;
```



If you want to completely replace the default error handling code, use the `evalException.message` variable in your code to return information for the exception.

## Insert Anything into Report?

Inserts text only if you select one of the following options:

- **Insert MATLAB expression string in report**
- **Display command window output in report**

## Class

`rptgen.cml_eval`

## See Also

Insert Variable, MATLAB Property Table, MATLAB Toolbox Version Number, Variable Table

## Figure Loop

Apply child components to specified graphics figures

### Description

This component applies each child component to specified figures in the report. For more information about working with this component, see “Logical and Looping Components” on page 6-17.

### Figure Selection

- **Include figures**
  - **Current figure only:** Includes only the current figure in the report.
  - **Visible figures:** Loops on all visible figures. The **Data figures only** option is checked by default and excludes figures with `HandleVisibility = 'off'` from the loop.
  - **All figures with tags:** Loops on figures with specified tags, select When you select a given tag, all figures with that tag appear in the loop, regardless of whether each figure is visible or whether its `HandleVisibility` attribute is 'on' or 'off'.

The tag field (located under **All figures with tags**) shows selected tags. To add tags to this field, type in the tag names, separating them with new lines.

- **Loop Figure List:** Shows all figures that are included in the loop. If the report setup file generates new figures or changes existing figures, figures in the **Loop Figure List** are not the figures that are reported on.

### Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each object found in the loop.
- **Display the object type in the section title:** Inserts the object type automatically into the section title in the generated report.
- **Create link anchor for each object in loop:** Create a link target on each object in the loop so that other parts of the report can link to it.

### Insert Anything into Report?

Yes, inserts a section if you select the **Create section for each object in loop** option.

### Class

`rptgen_hg.chg_fig_loop`

**See Also**

Axes Loop, Axes Snapshot, Figure Snapshot, Graphics Object Loop, Handle Graphics Linking Anchor, Handle Graphics Name, Handle Graphics Parameter, Handle Graphics Property Table, Handle Graphics Summary Table

## Figure Snapshot

Insert snapshot of Handle Graphics figure into report

### Description

This component inserts a snapshot of a Handle Graphics figure into the report.

### Format

- **Image file format:** Specifies the image file format. Select Automatic HG Format to automatically choose the format best suited for the specified report output format. Otherwise, choose an image format that your output viewer can read. Automatic HG Format is the default option. Other options include:
  - Automatic HG Format (uses the Handle Graphics file format selected in the Preferences dialog box)
  - Bitmap (16m-color)
  - Bitmap (256-color)
  - Black and white encapsulated PostScript
  - Black and white encapsulated PostScript (TIFF)
  - Black and white encapsulated PostScript2
  - Black and white encapsulated PostScript2 (TIFF)
  - Black and white PostScript
  - Black and white PostScript2
  - Color encapsulated PostScript
  - Color encapsulated PostScript (TIFF)
  - Color encapsulated PostScript2
  - Color encapsulated PostScript2 (TIFF)
  - Color PostScript
  - Color PostScript2
  - JPEG high quality image
  - JPEG medium quality image
  - JPEG low quality image
  - PDF image
  - PNG 24-bit image
  - TIFF - compressed
  - TIFF - uncompressed
  - Windows metafile
- **Capture picture from screen:**
  - Client area only: Captures a portion of the figure window.

- Entire figure window: Captures the entire figure window.

## Print Options

- **Paper orientation:**

- Landscape
- Portrait
- Rotated
- Use figure orientation: Uses the orientation for the figure, which you set with the `orient` command.
- Full page image (PDF only): In PDF reports, scales images to fit the full page, minimizes page margins, and maximizes the size of the image by using either a portrait or landscape orientation.

For more information about paper orientation, see the `orient` command in the MATLAB documentation.

- **Image size:**

- Use figure `PaperPositionMode` setting: Uses the figure's `PaperPositionMode` property as the image size in the report. For more information about paper position mode, see the `orient` command in the MATLAB documentation.
- Automatic (same size as on screen): Sets the image in the report to the same size as it appears on the screen.
- Custom: Specifies a custom image size. Set the image size in the **Size** field and **Units** list.
- **Size:** Specifies the size of the figure snapshot in the form `w h` (width times height). This field is active only if you choose `Custom` from the **Image size** selection list.
- **Units:** Specifies units for the size of the figure snapshot. This field is active only if you choose `Custom` in the **Image size** selection list.
- **Invert hardcopy:** Sets print colors using the figure's `InvertHardcopy` property, which inverts colors for printing. Options include:
  - Automatic: Automatically changes dark axes colors to light axes colors. If the axes color is a light color, it is not inverted.
  - Invert: Changes dark axes colors to light axes colors and vice versa.
  - Don't invert: Does not change the colors in the image.
  - Use figure's `InvertHardcopy` setting: Uses the value of the `InvertHardcopy` property set in the Handle Graphics image.
  - Make figure background transparent: Makes the image background transparent.

## Display Options

- **Scaling:**

- Fixed size: Specifies the number and type of units.
- Zoom: Specifies the percentage, maximum size, and units of measure.
- Use image size: Causes the size of the image in the report to appear the same size as on screen.

- **Size:** Specifies the size of the snapshot in the format w h (width, height). This field is active only if you choose **Fixed size** in the **Scaling** list.
- **Max size:** Specifies the maximum size of the snapshot in the format w h (width, height). This field is active only if you choose **Zoom** from the **Scaling** list.
- **Units:** Specifies units for the size of the snapshot. This field is active only if you choose **Zoom** or **Fixed size** in the **Image size** selection list.
- **Alignment** Only reports in PDF or RTF format support this property. Options include:
  - Auto
  - Right
  - Left
  - Center
- **Title:** Specifies a title for the figure:
  - Custom: Specifies a custom title.
  - Name: Specifies the figure name as the title.
- **Caption:** Specifies text to appear under the snapshot.

## Insert Anything into Report?

Yes. Image.

### Class

rptgen\_hg.chg\_fig\_snap

### See Also

Axes Loop, Axes Snapshot, Figure Loop, Graphics Object Loop, Handle Graphics Linking Anchor, Handle Graphics Name, Handle Graphics Parameter, Handle Graphics Property Table, Handle Graphics Summary Table

# For Loop

Iteratively execute child components

## Description

This component functions like the MATLAB `for` loop, except that instead of executing a statement, it executes its child components. It must have at least one child component to execute.

## Loop Type

The loop type can have incremented indices or a vector of indices. For more information on for loops and indices, see `for`.

- **Incremented indices:** Executes a `for` loop of the form:

```
for varname=x:y:z
```

- **Start:** Corresponds to `x` in the previous expression.
- **Increment:** Corresponds to `y` in the previous expression.
- **End:** Corresponds to `z` in the previous expression.
- **Vector of Indices:** Executes a `for` loop of the form:

```
for varname=[a b c ...]
```

Specify appropriate values in the **Vector** field in the form `a b c ...`

## Workspace Variable

- **Show index value in base workspace:** Displays the loop index in the MATLAB workspace while other components execute.
- **Variable name:** Allows you to specify the variable name. The default is `RPTGEN_LOOP`.
- **Remove variable from workspace when done:** Removes the loop index from the MATLAB workspace. This option is only available if you select the **Show index value in base workspace** option.

## Insert Anything into Report?

No.

## Class

```
rptgen_lo.clo_for
```

## See Also

Logical Else, Logical Elseif, Logical If, Logical Then, While Loop

## Graphics Object Loop

Run child components for each Handle Graphics object open in MATLAB workspace

### Description

This component runs its child components for each Handle Graphics object that is currently open in the MATLAB workspace. The component inserts a table into the generated report.

### Select Objects

- **Exclude GUI objects (uicontrol, uimenu, ...):** Excludes graphical interface objects, such as `uicontrol` and `uimenu`, from the loop.
- **Loop list:** Specifies the loop level for Handle Graphics objects:
  - Loop on objects with handle visibility "on"
  - Loop on all objects
  - **Search for:** Allows you to enter space-delimited search terms.

### Section Options

- **Create section for each object in loop:** Inserts a section in the generated report for each object found in the loop.
- **Display the object type in the section title:** Inserts the object type automatically into the section title in the generated report.
- **Create link anchor for each object in loop:** Create a link target on each object in the loop so that other parts of the report can link to it.

### Insert Anything into Report?

Yes, inserts a section if you select the **Create section for each object in loop** option.

### Class

`rptgen_hg.chg_obj_loop`

### See Also

Axes Loop, Axes Snapshot, Figure Loop, Figure Snapshot, Handle Graphics Linking Anchor, Handle Graphics Name, Handle Graphics Parameter, Handle Graphics Property Table, Handle Graphics Summary Table



# Handle Graphics Linking Anchor

Designate location to which links point

## Description

This component designates a location to which links point. It should have a looping component as its parent.

## Properties

- **Insert text:** Specifies text to appear after the linking anchor.
- **Link from current:** Sets the current model, system, block, or signal as the linking anchor:
  - **Automatic:** Automatically selects the appropriate figure, axes, or object as a linking anchor. If the `Figure Loop` component is this component's parent, the linking anchor points to the current figure. Similarly, if the `Graphics Object Loop` is this component's parent, the linking anchor points to the current object.
  - **Figure:** Sets the linking anchor to the current figure.
  - **Axes:** Sets the linking anchor to the current axes.
  - **Object:** Sets the linking anchor to the current object.

## Insert Anything into Report?

Yes. Anchor.

## Class

`rptgen_hg.chg_obj_anchor`

## See Also

`Axes Loop`, `Axes Snapshot`, `Figure Loop`, `Figure Snapshot`, `Graphics Object Loop`, `Handle Graphics Name`, `Handle Graphics Parameter`, `Handle Graphics Property Table`, `Handle Graphics Summary Table`

## Handle Graphics Name

Insert name of Handle Graphics object into the report

### Description

This component inserts the name of a Handle Graphics object as text into the report. You can use this component to create a section title based on the current figure by making it the first child component of a **Chapter/Subsection** component, and then selecting the **Chapter/Subsection** component's **Get title from first child component** option.

### Properties

- **Display name as:**
  - Type Name
  - Type –Name
  - Type: Name
- **Show name of current:**
  - **Figure:** Shows the name of the current figure. The first nonempty figure parameter determines the name.
  - **Axes:** Shows the name of the current axes. The first nonempty axes parameter determines the name.
  - **Other Object:** Sets the name of the current object to the figure's `CurrentObject` parameter and its first nonempty parameter.

### Insert Anything into Report?

Yes. Text.

### Class

`rptgen_hg.chg_obj_name`

### See Also

Axes Loop, Axes Snapshot, Figure Loop, Figure Snapshot, Graphics Object Loop, Handle Graphics Linking Anchor, Handle Graphics Parameter, Handle Graphics Property Table, Handle Graphics Summary Table

# Handle Graphics Parameter

Insert property name/property value pair from Handle Graphics figure, axes, or other object

## Description

This component inserts a property name/property value pair from a Handle Graphics figure, axes, or other object.

## Property Selection

- **Get property from current:** Reports on a specified Handle Graphics object:
  - **Figure:** Inserts a figure's property name/property value pair.
  - **Axes:** Inserts an axes' property name/property value pair.
  - **Object:** Inserts an object's property name/property value pair.
- **Figure property:** Specifies the type of property to include. The **All** option shows every parameter for the current object.

## Display Options

- **Title:** Specifies a title for the generated report:
  - **None** (default): No title.
  - **Automatic:** Automatically generates the title from the parameter.
  - **Custom** Specifies a custom title.
- **Size limit:** Limits the width of the display in the generated report. Units are in pixels. The size limit of a given table is the hypotenuse of the table width and height [ $\sqrt{w^2+h^2}$ ]. The size limit of a text equals its number of characters squared. If you exceed the size limit, the variable appears in condensed form, such as [64x64 double]. Setting a size limit of 0 displays the variable in full, no matter how large it is.
- **Display as:** Choose a display style:
  - **Auto table/paragraph:** Displays as a table or paragraph.
  - **Inline text:** Displays in line with the surrounding text.
  - **Paragraph:** Displays as a paragraph.
  - **Table:** Displays as a table.
- **Ignore if value is empty:** Excludes empty parameters from the generated report.

## Insert Anything into Report?

Yes. Text.

## Class

rptgen\_hg.chg\_property

## **See Also**

Axes Loop, Axes Snapshot, Figure Loop, Figure Snapshot, Graphics Object Loop, Handle Graphics Linking Anchor, Handle Graphics Name, Handle Graphics Property Table, Handle Graphics Summary Table

# Handle Graphics Property Table

Insert table that reports on property name/property value pairs

## Description

This component inserts a table that reports on property name/property value pairs.

For more information on using this component, see “Property Table Components” on page 6-6.

## Select Graphics Object

- **Object type:** Specifies an object type for the generated report:
  - **Figure**
  - **Axes**
  - **Object**
- **Filter by class:** Specifies a class or classes for the table.

## Table

You can select a preset table, which is already formatted and set up, from the list in the upper-left corner of the attributes page.

To create a custom table, edit a preset table, such as **Blank 4x4**. Add and delete rows and add properties. To open the Edit Table dialog box, click **Edit**.

For details about creating custom property tables, see “Property Table Components” on page 6-6.

- **Preset table:** Specifies the type of table to display the object property table.
  - **Defaults**
  - **Callbacks**
  - **Graphics**
  - **Printing**
  - **Blank 4x4**

To apply a preset table, select the table and click **Apply**.

- **Split property/value cells:** Splits property name/property value pairs into separate cells. For the property name and property value to appear in adjacent horizontal cells in the table, select the **Split property/value cells** check box. In this case, the table is in split mode and there can be only one property name/property value pair in a cell. If there is more than one name/property pair in a cell, only the first pair appears in the report. All subsequent pairs are ignored.

For the property name and property value to appear together in one cell, clear the **Split property/value cells** check box. In this case, the table is in nonsplit mode, which supports more than one property name/property value pair per cell. It also supports text.

Before switching from nonsplit mode to split mode, make sure that you have only one property name/property value pair per table cell. If you have more than one property name/property value pair or any text, only the first property name/property value pair appears in the report; subsequent pairs and text are omitted.

- **Display outer border:** Displays the outer border of the table in the generated report.

## Table Cells

Select table properties to modify. The selection in this pane affects the availability of fields in the **Title Properties** pane.

If you select **Figure Properties**, only the **Contents** and **Show** options appear. If you select any other object in the **Table Cells** pane, the **Lower border** and **Right border** options appear.

## Title Properties

- **Contents:** Enables you to modify the contents of the table cell selected in the **Table Cells** pane. Options include:
  - **Left**
  - **Center**
  - **Right**
  - **Double justified**
- **Show as:** Enables you to choose the format for the contents of the table cell. Options include:
  - **Value**
  - **Property Value**
  - **PROPERTY Value**
  - **Property: Value**
  - **PROPERTY: Value**
  - **Property - Value**
  - **PROPERTY - Value**
- **Alignment:** Aligns text in the table cells. Options are:
  - **Left**
  - **Center**
  - **Right**
  - **Double-justified**
- **Lower border:** Displays the lower border of the table in the generated report.
- **Right border:** Displays the right border of the table in the generated report.

## Insert Anything into Report?

Yes. Table.

**Class**

rptgen\_hg.chg\_prop\_table

**See Also**

Axes Loop, Axes Snapshot, Figure Loop, Figure Snapshot, Graphics Object Loop, Handle Graphics Linking Anchor, Handle Graphics Name, Handle Graphics Parameter, Handle Graphics Summary Table

## Handle Graphics Summary Table

Insert table that summarizes Handle Graphics object properties

### Description

This component inserts a table that summarizes Handle Graphics object properties. Each row in the table represents an object. Each column in the table represents a property. You can specify object properties to include in the report.

### Properties

- **Object type:** Specifies the object type to display in the generated report. Options include:
  - figure
  - axes
  - object

The available options in the **Select Objects** pane depend on your selection in the **Object type** menu.

- **Table title:** Specifies a title for the table in the generated report. Options include:
  - Automatic: Generates a title automatically.
  - Custom: Specifies a custom title.

### Property Columns

- **Property columns:** Specifies object properties to include in the table in the generated report.
  - To add a property:
    - 1 Select the appropriate property level in the menu.
    - 2 In the list under the menu, select the property to add and click **Add**.
  - To delete a property, select its name and click **Delete**.

Some entries in the list of available properties (such as **Depth**) are “virtual” properties which you cannot access using the `get_param` command. The properties used for property/value filtering in the block and system loop components must be retrievable by the `get_param`. Therefore, you cannot configure your summary table to report on all blocks of `Depth == 2`.

- **Remove empty columns:** Removes empty columns from the table in the generated report.
- **Transpose table:** Changes the summary table rows into columns in the generated report, putting the property names in the first column and the values in the other columns.

### Object Rows

**Insert anchor for each row:** Inserts an anchor for each row in the summary table.



## Figure Selection

The options displayed in the **Figure Selection** pane depend on the object type selected in the **Object type** list:

- If **Object type** is figure, the following options appear:
  - **Include figures**
    - **Current figure only:** Includes only the current figure in the report.
    - **Visible figures:** Executes child components for figures that are currently open and visible. The **Data figures only** option is checked by default. This option excludes figures with `HandleVisibility = 'off'` from the loop.
    - **All figures with tags:** Includes all figures with a specified tag regardless of whether they are visible or their `HandleVisibility` parameter is 'on' or 'off'. The tag selection list, located under this option, shows available tags. You can add tag names to this list.
  - **Data figure only (Exclude applications):** Shows only data figures.
  - **Loop Figure List:** Shows figures within the current set of figures to display.
- If **Object type** is axes, the following options appear:
  - **Loop type:**
    - **All axes:** Loops on all axes objects.
    - **Current axes:** Loops on the selected axes object.
  - **Exclude objects which subclass axes:** Excludes objects such as legends and color bars.
  - **Loop Menu:**
    - **Loop on axes with handle visibility "on":** Loops on visible axes objects.
    - **Loop on all axes:** Loops on all axes objects.
  - **Search terms:** Specifies search terms for the loop. For example, to search for Tag and My Data, enter "Tag", "My Data".
- If **Object type** is object, the following options appear:
  - **Exclude GUI objects (uicontrol, uimenu, ...):** Excludes graphical interface objects, such as `uicontrol` and `uimenu`, from the loop.
  - **Loop menu:** Specifies the loop level:
    - **Loop on objects with handle visibility "on"**
    - **Loop on all objects**
  - **Search for:** Specifies space-delimited search terms.

## Insert Anything into Report?

Yes. Table.

## Class

`rptgen_hg.chg_summ_table`

## **See Also**

Axes Loop, Axes Snapshot, Figure Loop, Figure Snapshot, Graphics Object Loop, Handle Graphics Linking Anchor, Handle Graphics Name, Handle Graphics Parameter, Handle Graphics Property Table

# HTML Text

Insert HTML-formatted text into reports generated from a template

## Description

This component inserts text formatted with HTML markup into the report. You can use this component only if you set the report's **File format** to one of the `from template` options, such as `Direct PDF (from template)`. The **HTML Text** component can have the **Chapter/Subsection** component or the report as its parent.

For Word or PDF report output, the text is converted to Word or PDF and formatted according to the HTML markup. You can enter the HTML text directly in the component or specify a file or workspace variable that contains the text.

## HTML Text Options

- **HTML source:** Specifies the source of the HTML text.
- **HTML string:** If you select `String` as the HTML source, enter the HTML text in the text box.
- **File name:** If you select `File` as the HTML source, specify the `.htm` or `.html` file that contains the HTML text. Click the file browser button to select a file from a folder.
- **Workspace variable name:** If you select `Workspace variable` as the HTML source, specify the name of the variable that contains the HTML text.

## Insert Anything into Report?

Yes. Text formatted according to the HTML markup.

## Class

`rptgen.cfr_html_text`

## See Also

Chapter/Subsection

## Image

Insert image from external file into report

### Description

This component inserts an image from an external file into the report. It can have the **Chapter/Subsection** or **Paragraph** component as its parent. If the **Paragraph** component is its parent, you must select the **Insert as inline image** check box.

### Class

- **File name:** Specifies the image file name. You can enter this name manually, or use the **Browse** button (...) to find the image file.

The image must be in a format that your viewer can read. An error like the following appears if you specify the name of an image file that does not exist:

```
No file name. Could not create graphic.
```

This field supports %<VariableName> notation.

- **Copy to local report files directory:** Saves a copy of the image to a local report files folder.

### Display Options

- **Scaling:** Controls size of the image, as displayed in a browser. Making an image larger using this option does not affect the storage size of the image, but the quality of the displayed image may decrease as you increase or decrease the size of the displayed image.

Generally, to achieve the best and most predictable display results, use the default setting of **Use image size**.

- **Use image size:** Causes the image to appear the same size in the report as on screen (default).
- **Fixed size:** Specifies the number and type of units.
- **Zoom:** Specifies the percentage, maximum size, and units of measure.
- **Size:** Specifies the size of the snapshot in the format w h (width, height). This field is active only if you choose **Fixed size** from the **Scaling** list.
- **Max size:** Specifies the maximum size of the snapshot in the format w h (width, height). This field is active only if you choose **Zoom** from the **Scaling** list;
- **Units:** Specifies units for the size of the snapshot. This field is active only if you choose **Zoom** or **Fixed size** in the **Image size** selection list.
- **Alignment:** Only reports in PDF or RTF formats support this format property. Options are:
  - Auto
  - Right
  - Left

- Center
- **Title:** Specifies text to appear above the snapshot.
- **Caption:** Specifies text to appear under the snapshot.
- **Full page image (\*PDF only):** In PDF reports, scales images to fit the full page, minimizes page margins, and maximizes the size of the image by using either a portrait or landscape orientation.

## Preview

The image that you specify in the **Image file name** field appears in this pane. You cannot preview Adobe PostScript images, or images with formats that the `imread` function does not support, such as `gif`.

Clicking an image causes it to display in full size.

## Insert Anything into Report?

Yes. Image.

## Class

`rptgen.cfr_image`

## See Also

Chapter/Subsection, Empty Component, List, Paragraph, Table, Text, Title Page

## Import File

Import ASCII text file into report

### Description

This component imports an ASCII text file into the report.

### Properties

- **File name:** Specifies the name of the file to import into the text field. You can enter a name, or use the **Browse** button (...) to find the file.

To use a MATLAB workspace variable in a path name, use the notation %<VariableName>. Examples:

- %<doc>
- %<dir>/%<doc>
- %<[dir, '/filename.txt']>
- %<dir>/filename.txt
- **Import file as:** Specifies formatting for the imported file. Options include:
  - **Plain text (ignore line breaks):** Imports the file as plain text without any line breaks (no paragraphs). If you select this option, the **Import File** component acts like the **Text** component, so it should have the **Paragraph** component as its parent.

The examples in this section use the following text as the input file:

```
This is the first row of text from the imported file.  
The second row follows a line break in the first row.
```

```
There is a blank line above the third row.
```

Plain text (ignore line breaks) produces the following formatting for the example file:

```
This is the first row of text from the imported file.  
The second row follows a line break in the first row.
```

```
There is a blank line above the third row.
```

- **Paragraphs defined by line breaks:** Imports the file as text, in paragraphs with line breaks (hard returns or carriage returns). This option produces the following formatting for the example file:

```
This is the first row of text from the imported file.  
The second row follows a line break in the first row.
```

```
There is a blank line above the third row.
```

- **Paragraphs defined by empty rows:** Imports the file as text, in paragraphs with empty rows (rows that include no text). This option produces the following formatting for the example file:

This is the first row of text from the imported file.  
The second row follows a line break in the first row.

There is a blank line above the third row.

- **Text (retain line breaks)** (default): Imports the file as plain text with line breaks. This option produces the following formatting for the example file:

This is the first row of text from the imported file.  
The second row follows a line break in the first row.

There is a blank line above the third row.

- **Fixed-width text (retain line breaks)**: Imports the file as fixed-width text (all letters have the same width or size), including line breaks. This option is useful for importing MATLAB files. This option produces the following formatting for the example file:

This is the first row of text from the imported file.  
The second row follows a line break in the first row.

There is a blank line above the third row.

- **DocBook XML**: Inserts an XML source file, and makes no changes to its format.
- **Formatted Text (RTF/HTML)**: Inserts an RTF or HTML source file, and makes no changes to its format.
- **Syntax highlighted MATLAB code**: Inserts a MATLAB file.

The **File Contents** field displays the first few lines of the file to be imported.

## Insert Anything into Report?

Yes.

- Inserts text if you select one of the following options:
  - Plain text (ignore line breaks)
  - Text (retain line breaks)
  - Fixed-width text (retain line breaks)
- Inserts paragraphs if you select one of the following options:
  - Paragraphs defined by line breaks
  - Paragraphs defined by empty rows
- Inserts the contents of an XML file if you select the **DocBook XML** option.
- Inserts the contents of the RTF or HTML file if you select the **Formatted text (RTF/HTML)** option.
- Inserts a link to a file if you import the file into an HTML report.

## Class

rptgen.crg\_import\_file

**See Also**

Comment, Nest Setup File, Stop Report Generation, Time/Date Stamp



# Insert Variable

Insert variable values into report

## Description

This component inserts the value (and, optionally, the name) of each the following variables into the report:

- A variable from the MATLAB workspace
- A variable from a MAT-file
- A global variable
- A variable that you specify directly

## Source

- **Variable name:** Specifies the name of the variable:
  - `%<VariableName>`: Inserts the value of a variable from the MATLAB workspace into the report.
- **Variable location:**
  - `Base Workspace`: Gets a variable from the MATLAB workspace.
  - `MAT File`: Gets a variable from a binary file with a `.mat` extension.
  - `Global variable`: Gets a global variable.
  - `Direct`: Gets a variable that you specify directly.

## Display Options

- **Title:** Specify a title for the report:
  - `Automatic`: Generates a title automatically.
  - `Custom`: Specifies a custom title.
  - `None`: Specifies no title.
- **Array size limit:** Limits the width of the display in the generated report. Units are in pixels. The size limit for a given table is the hypotenuse of the table width and height [ $\sqrt{w^2+h^2}$ ]. The size limit for text is the number of characters squared. If you exceed the size limit, the variable appears in condensed form, such as `[64x64 double]`. Setting a size limit of `0` displays the variable in full, regardless of its size.
- **Object depth limit:** Specifies the maximum number of nesting levels to report on for a variable value
- **Object count limit:** Specifies the maximum number of nested objects to report on for a variable value
- **Display as:** Choose a display style from the menu:
  - `Table`: Displays as a table.

- Paragraph: Displays as a paragraph.
- Inline text: Displays inline with the surrounding text.
- Table or paragraph depending on data type: Displays as a table or paragraph.
- **Show variable type in headings:** Show data type of this variable in the title of its report.
- **Show variable table grids:** Show grid lines for the table used to report the value of this variable.
- **Make variable table page wide:** Make the variable table as wide as the page on which the table appears.
- **Omit if value is empty:** Exclude empty parameters from the generated report.
- **Omit if property default value:** Exclude object property from the report if that property uses the default value.

## Insert Anything into Report?

Yes. Text.

## Class

rptgen.cml\_variable

## See Also

Evaluate MATLAB Expression, MATLAB Property Table, MATLAB Toolbox Version Number, Variable Table

# Line Break

Insert a line break

## Description

This component inserts a line break into a report. You can insert a line break as a child of the **Document**, **Paragraph** or **Table Entry** component and as a sibling of a Text component.

## Insert Anything into Report?

Yes. Line break.

## Class

`rptgen.cfr_line_break`

## See Also

Paragraph, Table, Text

# Link

Insert linking anchors or pointers into report

## Description

This component inserts linking anchors or pointers into the report.

For a PDF report, if you open the report from MATLAB (for example, if you open the report right after generating it), the link does not work. However, if you open a PDF report outside of MATLAB (for example, from Adobe Acrobat), the link works properly.

## Properties

- **Link type:** Select the type of link to insert into the report. Options include:
  - **Linking anchor:** Specifies a link to an anchor.
  - **Internal document link:** Specifies a location in the report (as specified by an anchor).
  - **URL (external) link:** Specifies a link to a Web site or to a MATLAB command to execute from generated report.
- **Link identifier:** Indicates the location to which the link points. It can contain only ASCII characters, and it is not visible in the generated report.

For a Web link, the link identifier options are context sensitive; their formats differ depending on the link type you select. For example, to link to an external file `foo.txt`, specify the link identifier as follows:

- On UNIX systems:  
`file:///home/janedoe/foo.txt`
- On Microsoft Windows systems:  
`H:\foo.txt`

For a link to a MATLAB command, enter `matlab:` followed by a space and the MATLAB command that you want the link to execute.

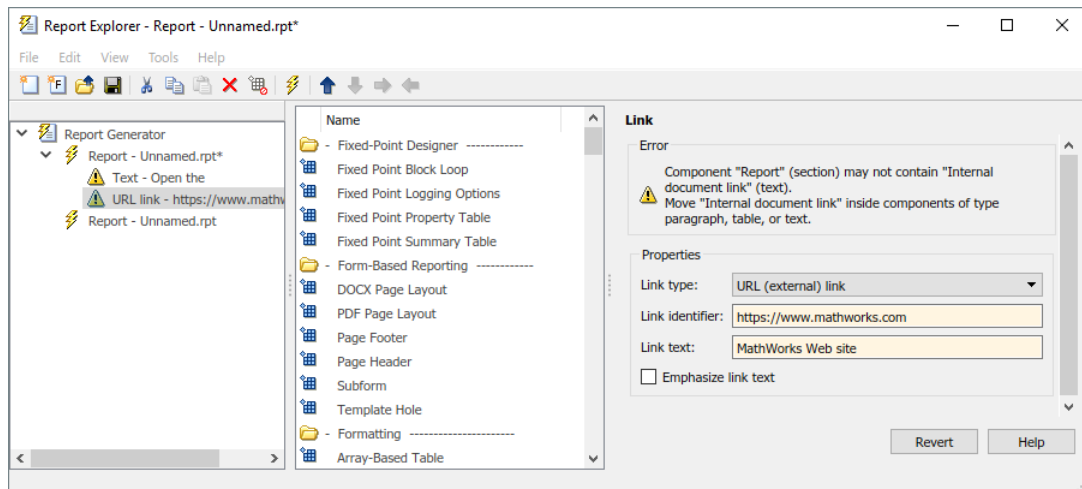
- **Link text:** Specifies text to use in the link.
- **Emphasize link text:** Italicizes the link text.

## Examples

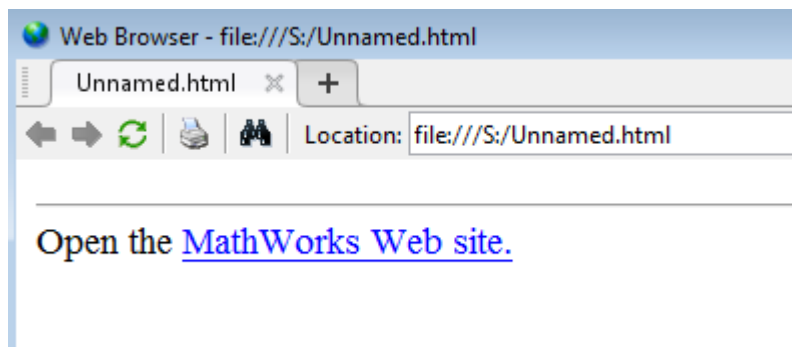
### Link to an External Web Site

- 1 Open Report Explorer with the `setedit` command.
- 2 In the Properties pane on the right, click **Create and edit a new Report file**.
- 3 In the Library pane in the middle, under the Formatting category, select the Text component and click the **Add component** icon.
- 4 In the Properties pane, enter `Open the` (add a blank space at the end of the text).

- 5 In the Library pane, under the Formatting category, select the Link component and click the **Add component** icon.
- 6 In the Properties pane:
  - Set **Link type** to URL (external) link.
  - In **Link Identifier**, enter `https://www.mathworks.com`.
  - In **Link text**, enter MathWorks Web site.



- 7 Generate the report.



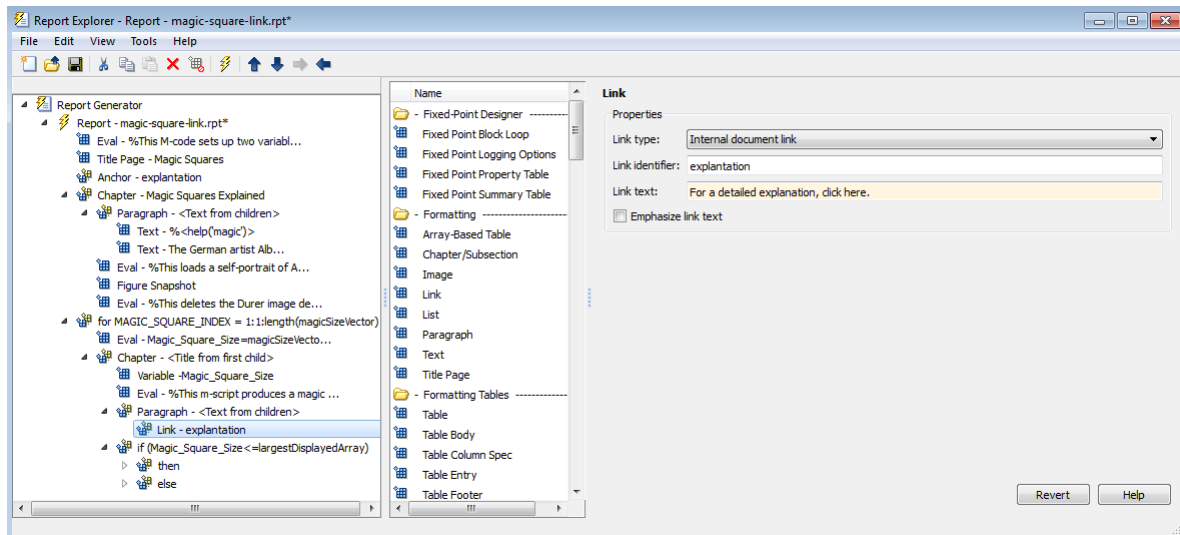
- 8 Click the link to open the MathWorks® website.

### Link to Another Place in a Report

- 1 At the MATLAB command line, enter `setedit magic-square.rpt`.
- 2 In the Outline pane on the left, select the **Title Page** component.
- 3 In the Library pane in the middle, under the Formatting category, select the Link component and click the **Add component** icon.
- 4 In the Properties pane:
  - Set **Link type** to Linking anchor.
  - In **Link identifier**, enter explanation.

In the Contents pane, the Link component appears as Anchor - explanation.

- 5 In the Outline pane, under the second **Chapter** component, select the **Eval** component.
- 6 In the Library pane, under the Formatting category, select the **Paragraph** component and click the **Add component** icon.
- 7 In the Library pane, under the Formatting category, select the **Link** component and click the **Add component** icon.
- 8 In the Properties pane:
  - Set **Link type** to Internal document link.
  - In **Link identifier**, enter explanation.
  - In **Link text**, enter For a detailed explanation, click here.



- 9 Generate the report.

## Chapter 2. *Magic\_Square\_Size 4*

[For a detailed explanation, click here.](#)

16	2	3	13
5	11	10	8
9	7	6	12
4	14	15	1

## Chapter 3. *Magic\_Square\_Size 8*

[For a detailed explanation, click here.](#)

64	2	3	61	60	6	7	57
9	55	54	12	13	51	50	16
17	47	46	20	21	43	42	24
40	26	27	37	36	30	31	33
32	34	35	29	28	38	39	25
41	23	22	44	45	19	18	48
49	15	14	52	53	11	10	56
8	58	59	5	4	62	63	1

- 10 Click the link to move to near the top of the report, to “Chapter 1. Magic Squares Explained.”

### Link to a Model

This example shows how to add a link from a report to a Simulink model. To create and run this report, you must have Simulink installed.

- 1 Open Report Explorer with the `setedit` command.
- 2 In the report library, under `rptgennextdemos`, select `simulink-default.rpt` and open it.
- 3 From the component library, under Formatting, select the **Paragraph** component and add it to your report setup file after the Model Loop **Chapter** component.
- 4 Add a **Link** component after the new **Paragraph** component. Set these properties:
  - Set **Link type** to URL (external) link.
  - In **Link Identifier**, enter `matlab:open_system('%<RptgenSL.getReportedModel()>')`.
  - In **Link text**, enter Open model.

- 5 Generate the report.

### List of Tables

- 1.1. [f14 Simulation Parameters](#)
- 1.2. [Model Variables](#)
- 1.3. [Signal Properties](#)
- 1.4. [f14 System Information](#)
- 1.5. [Block Type Count](#)

## Chapter 1. f14

### Table of Contents

[f14](#)

[Open model](#)

- 6 To open the model from the report, click the link.

### Insert Anything into Report?

Yes. Text or anchor.

### Class

rptgen.cfr\_link

### See Also

Chapter/Subsection, Empty Component, List, Paragraph, Table, Text, Title Page



# List

Create bulleted or numbered list from cell array or child components

## Description

This component creates a bulleted or numbered list from a cell array or child components.

## List Content

- **Create list from workspace cell array:** Creates the list from of the 1-by-n or n-by-1 cell array. This option is not available when this component has child components — in this case, the list automatically generates from the child components.
- **List title:** Specifies the title of the list.
- **List title style name:** Specifies the style to use with the list title. To specify a style:
  - 1 Set the report's **File format** to one of the from `template` options, for example, `Direct PDF (from template)`.
  - 2 In the List properties dialog box, set **List title style name** to `Specify`.
  - 3 In the **List title style name** text box, type a style name.

To take effect, the style you specify must be a list style defined in the template that you use to generate the report. For more information about template styles, see "Report Templates" on page 7-2.

## List Formatting

- **List type:**
  - Bulleted list
  - Numbered list
- **Numbering style:** Specifies a numbering style for numbered lists. This setting is supported only in the RTF/DOC report format. Options include:
  - 1,2,3,4,...
  - a,b,c,d,...
  - A,B,C,D,...
  - i,ii,iii,iv,...
  - I,II,III,IV,...
- **List style name:** Specifies the style to use with the list. To specify a style:
  - 1 Set the report's **File format** to one of the from `template` options, for example, `Direct PDF (from template)`.
  - 2 Set **List style name** to `Specify`.
  - 3 In the **List style name** text box, type a style name.

To take effect, the style you specify must be defined in the template that you use to generate the report.

- **Show parent number in nested list (1.1.a)**: Displays all level numbers in a nested list. You can create a nested list by putting one cell array inside another or by nesting one **List** component inside another. Following is an example of how a list appears when you select this option:

1. Example
2. Example
  - 2.1. Example
  - 2.2. Example
    - 2.2.a. Example
    - 2.2.b. Example
3. Example

This option is not available if you select **Show only current list value (a)**.

- **Show only current list value (a)**: Displays only the current list value. Following is an example of how a list appears when you select this option:

1. Example
2. Example
  1. Example
  2. Example
    1. Example
    2. Example
3. Example

This option is not available if you select **Show parent number in nested list (1.1.a)**.

### Example 1: Creating a Nested List

Consider the following report setup file, which includes a nested list created by putting a **List** component inside another **List** component:

```
[ - ] Report - Unnamed.rpt
  [ - ] Bulleted list from child components
    [ ] Text - sky
    [ ] Table - varname
    [ ] Image - test
    [ ] Text - grass
  [ - ] Bulleted list from child components
    [ ] Text - clouds
    [ ] Text - sun
  [ - ] Paragraph - information
```

This report setup file generates a report that includes the following bulleted lists:

- sky
- varname, the table from the variable
- test, a snapshot of the image
- grass
  - clouds
  - sun

- information

### **Example 2: Creating a List Using Child Components**

To generate a report that includes the following bulleted list:

- red
- green
- blue

Use the following report setup file:

```
[ - ] Report - Unnamed.rpt
  [ - ] Bulleted list from child components
    [ ] Text - red
    [ ] Text - green
    [ ] Text - blue
```

### **Creating a List Using a Cell Array**

To generate the same bulleted list as in the previous example, configure a report setup file to call a cell array, `colors`:

```
[ - ] Report - Unnamed.rpt
  [ - ] Bulleted list from cell array called colors
```

Where `colors` is:

```
colors={'red','green','blue'}
```

## **Insert Anything into Report?**

Yes. List.

## **Class**

`rptgen.cfr_list`

## **See Also**

Chapter/Subsection, Empty Component, Link, Paragraph, Table, Text, Title Page

## Logical Else

Specify an `else` condition for a `Logical If` component

### Description

This component acts as an `else` when it is the child of the `Logical If` component. You can specify this component in one of the following ways:

- `if`  
    `then`  
    `else`
- `if`  
    `then`  
    `elseif`  
    `elseif`  
    `.`  
    `.`  
    `.`  
    `else`

### Properties

**If component has no children, insert text:** Inserts specified text into your report when the `Logical Else` component has no child components. In this case, this component acts like the `Text` component.

### Insert Anything into Report?

Yes, when `if` or `elseif` statement is false.

### Class

`rptgen_lo.clo_else`

### See Also

For `Loop`, `Logical Elseif`, `Logical If`, `Logical Then`, `While Loop`

# Logical Elseif

Specify an `elseif` condition for a `Logical If` component

## Description

This component acts as an `elseif` when it is the child of the `Logical If` component. You must specify this component as follows:

```
if
  then
  elseif
  elseif
  .
  .
  .
  else
```

## Properties

- **Test expression:** Specifies a MATLAB expression to evaluate.
- **If component has no children, insert text:** Inserts the specified text into the report when the `Logical Elseif` component has no child components. In this case, this component acts like the `Text` component.

## Insert Anything into Report?

Yes, when parent `if` statement is false.

## Class

```
rptgen_lo.clo_else_if
```

## See Also

For `Loop`, `Logical Else`, `Logical If`, `Logical Then`, `While Loop`

## Logical If

Specify logical if condition

### Description

This component acts as a logical if; it can have the Logical Then, Logical Elseif, or Logical Else components as children components. This component executes its child components when the specified workspace expression is true. It displays specified text when it has no child components. You can specify this component as follows:

- if  
    then
- if  
    then  
    else
- if  
    then  
    elseif  
    elseif  
    .  
    .  
    .  
    else

### Properties

- **Test expression:** Specifies a MATLAB expression to evaluate.
- **If component has no children, insert text:** Inserts specified text into the report when the Logical If component has no children.

### Insert Anything into Report?

Depends on specified attribute values.

### Class

rptgen\_lo.clo\_if

### See Also

For Loop, Logical Else, Logical Elseif, Logical Then, While Loop

# Logical Then

Specify a then condition for a Logical If component

## Description

This component acts as a then when it is the child of the Logical If component. You can specify this component as follows:

- if  
    then
- if  
    then  
    else
- if  
    then  
    elseif  
    elseif  
    .  
    .  
    .  
    else

## Attributes

**If component has no children, insert text:** Inserts specified text into the report when the Logical Then component has no children. In this case, this component acts like the Text component.

## Insert Anything into Report?

Yes, when parent if statement is true.

## Class

rptgen\_lo.clo\_then

## See Also

For Loop, Logical Else, Logical Elseif, Logical If, While Loop

## MATLAB Property Table

Insert table that includes MATLAB object property name/property value pairs

### Description

This component inserts a table that includes MATLAB object property name/property value pairs.

### Table

Select a preset table, which is already formatted and set up, in the preset table list in the upper-left corner of the attributes page.

- **Preset table:** Choose a type of table:
  - Default
  - Blank 4x4

To apply the preset table, select the table and click **Apply**.

- **Split property/value cells:** Splits property name/property value pairs into separate cells. Select the **Split property/value cells** check box for the property name and property value to appear in adjacent cells. In this case, the table is in split mode; only one property name/property value pair per cell is allowed. If more than one name/property pair exists in a cell, only the first pair appears in the report; subsequent pairs are ignored.

Clear the **Split property/value cells** check box for a given property name and property value to appear together in one cell. In this case, the table is in nonsplit mode, which supports more than one property name/property value pair. It also supports text.

Before switching from nonsplit mode to split mode, make sure that you have only one property name/property value pair per table cell.

- **Display outer border:** Displays the outer border of the table in the generated report.
- **Table Cells:** Modifies table properties. The selection in this pane affects the available fields in the **Cell Properties** pane.

### Cell Properties

Available options in the **Cell Properties** pane depend what you select for **Table Cells**. If you select **Workspace Properties**, only the **Contents** and **Show** options appear. If you select any other option, the **Lower border** and **Right border** options appear.

- **Contents:** Modifies the contents of the table cell selected in the **Table Cells** pane.
- **Show as:** Specifies the format for the contents of the table cell. Options include:
  - Value
  - Property Value
  - PROPERTY Value
  - Property: Value



- PROPERTY: Value
- Property - Value
- PROPERTY - Value
- **Alignment:** Specifies how to align the contents of the selected table cell in the **Table Cells** field. Options include:
  - Left
  - Center
  - Right
  - Double justified
- **Lower border:** Displays the lower border of the table in the generated report.
- **Right border:** Displays the right border of the table in the generated report.

### Creating Custom Tables

To create a custom table, edit a preset table, such as Blank 4x4. You can add and delete rows and add properties. To open the Edit Table dialog box, click **Edit**.

For details about using this dialog box to create custom property tables, see “Property Table Components” on page 6-6.

## Insert Anything into Report?

Yes. Table.

### Class

rptgen.cml\_prop\_table

### See Also

Evaluate MATLAB Expression, Insert Variable, MATLAB Toolbox Version Number, Variable Table

## MATLAB Toolbox Version Number

Insert table that shows version and release numbers and release date of MathWorks products

### Description

Using the Table Filter, specify whether this component reports version information for all installed MathWorks products or just those products required for a model.

For the specified set of products, this component inserts a table showing any of these columns that you specify:

- Version number
- Release number
- Release date
- Is required for model

You can list all your MathWorks products by typing `ver` at the MATLAB command line.

### Table Title

**Table title:** Specifies the table title. The default is `version number`.

### Table Filter

**Show only toolboxes required for model:** When you select this option, the report shows version information for only those products required for a model or chart. By default, the report shows version information for all installed MathWorks products.

---

**Note** This option uses the Dependency Analyzer to determine what products appear in the version information table. See “Dependency Analyzer Scope and Limitations” (Simulink).

---

### Table Columns

- **Version number:** Includes the product version number (for example, 3.4) for all installed MathWorks products or for only those products required for a model or chart.

or

- **Release number:** Includes the MathWorks release number (for example, R2009b) for all installed MathWorks products or for only those products required for a model or chart.
- **Release date:** Includes the release date of for all installed MathWorks products or for only those products required for a model.
- **Is required for model:** Indicates “Yes” for each MathWorks product required for a model or chart.

## **Insert Anything into Report?**

Yes. Table.

### **Class**

rptgen.cml\_ver

### **See Also**

Evaluate MATLAB Expression, Insert Variable, MATLAB Property Table, Variable Table

## Nest Setup File

Allow one report setup file (rpt file) to run inside another

### Description

This component runs another report setup file at the point where the **Nest Setup File** component is located in the current report setup file.

The components of the inserted report setup file are stored in the current report setup file at the same level as the **Nest Setup File** component. Thus, inserted components have the same parent component as the **Nest Setup File** component.

### Properties

- **Report filename:** Specifies the name of the report setup file to import and run. You can enter a path to the file or use the **browse** button (...) to find the file. You can enter an absolute path or a relative path, relative to the report into which you nest the report.
- **Nest all reports with specified file name:** Nests all reports with the same name as specified in the **Report filename** option.
- **Inline nested report in this report:** Inserts the nested report in the original report setup file where this component is located.
- **Recursion limit:** Allows you to nest a report setup file inside itself by setting a recursion limit in this field. The recursion limit sets a limit on the number of times the report setup file can run itself.
- **Insert link to external report:** Creates two separate reports, one using the original report setup file and one using the nested report setup file. The report that includes the nested report includes an absolute path link to the nested report.
- **Link to external report is relative:** If you select **Insert link to external report**, then you can use the **Link to external report is relative** option to ensure the link to the nested report is a relative link. This feature facilitates including the report on a Web site.
- **Increment file name to avoid overwriting:** Appends a number to the file name of report that includes the nested report, to preserve earlier versions of current report file.

The Nest Setup File dialog box displays the report description of the nested report, if the nested report has a report description.

### Example

In the following example, the report setup file R2 is nested in R1:

```

[-] Report - R1.rpt
  [ ] Chapter
    [-] B
      [ ] Nest Setfile - R2.rpt
        [ ] C
          [ ] D
    [-] Report - R2.rpt
      [ ] 1
      [ ] 2
      [-] Chapter
        [ ] 4
        [ ] 5

```

The generated report is identical to the one generated by the following report setup file:

```
[ - ] Report - R1.rpt
  [ ] Chapter
  [ - ] B
    [ ] 1
    [ ] 2
    [ - ] Section 1
      [ ] 4
      [ ] 5
    [ ] C
  [ ] D
```

Components that determine their behavior from their parents, such as Chapter/Subsection, are affected by components in the parent report setup file.

## Insert Anything into Report?

Yes, if the nested report setup file produces a report.

## Class

rptgen.crg\_nest\_set

## See Also

Comment, Import File, Stop Report Generation, Time/Date Stamp

## Page Break

Insert a page break

### Description

This component inserts a page break into a paginated report. Paginated reports are Word, Direct PDF, and PDF (from Word) file formats. For form-based reports, you can insert page breaks in block type **Template Hole** components. For nonform-based reports, you can insert page breaks in **Chapter/Subsection** components. You can also insert page breaks in any **Logical and Flow Control** group component. A **Logical and Flow Control** group component must be a child of a **Template Hole** or **Chapter/Subsection**, for form-based or nonform-based reports, respectively.

### Insert Anything into Report?

Yes. Page break.

### Class

rptgen.cfr\_page\_break

### See Also

Chapter/Subsection, Line Break, Template Hole

## Page Footer

Page footer in a form-based PDF or Word report

### Description

Generates a page footer in a Word or PDF report. A template associated with this component defines its fixed content and format and holes for filling the footer with generated content. You can use this component to generate up to three types of footers per section: one for the first page, one for odd pages, and one for even pages.

You can define your report's page footers in templates. When you assign a template that defines footers to your report's Report Form or a Subform component, the Report Generator creates a **Page Footer** component for each page footer defined in each page layout defined by the template. It appends the page footer components that it creates for a particular template-defined page layout to the page layout component that it generates for that template-defined layout.

In addition to defining page footers in your templates, you can define them directly in the Report Explorer. For example, you can:

- Create page layout components in the Report Explorer and add footer components to them.
- Add footers to layouts defined in your templates.
- Change the templates assigned to footers defined in your report templates.

The Report Generator generates a **Template Hole** component for each hole defined by a page footer template. Every footer has at least one hole component, a **#start#** component. You can define additional holes in the template that defines the footer. You can add content to a footer by appending components to the footer holes in your report setup. The Report Generator generates footer content by executing its hole components when it generates the parent page layout. The generated content applies to all pages of the specified footer type. This means that you cannot use the Report Explorer to generate the footer content for specific pages. You can, however, use Word and PDF fields, such as page number fields, in footer templates to generate page numbers and other kinds of content that varies from page to page in a section.

To understand how layouts work, see "Define Page Layouts in a Form-Based Report Setup" on page 16-4.

### Page Footer Options

The options indicate the type of page the footer applies to and the template that defines the footer's form.

- **Page type:** The type of page in this page layout that the footer applies to.
  - **Default:** Footer for odd pages of the section, even pages if you do not specify an even-page footer, and the first page if you do not specify a first-page footer.
  - **Even:** Footer for even pages of the section.
  - **First:** Footer for the first page of the section.
- **Template type:** Specify the template that defines the footer content.

- **Library:** Select **Library** to select a template from a document part library. When you select this option, the **Source Library Options** appear.
- **File:** Select **File** to select a template file as the source of the footer content.
- **Page Layout:** This option appears if the **Page Footer** component was based on a footer in the template assigned to a **Report Form** or **Subform** component in your report setup.
- **Source Library Options:** If you select **Library** as the template type, you can set these options.
  - **Report form library:** Template library of the template file assigned to this report setup's **Report Form** component.
  - **Parent subform library:** Template library used by the **Subform** that contains this **Footer** component. This option appears only if this component is a descendant of a **Subform** component and the parent subform uses a library as the source of its template.
  - **Other library:** Template library of a specified template file.
- **Template:** If the template type is **File**, this option specifies the name of the template file that defines the footer associated with this component. If the template type is **Library**, this option specifies the template file that contains the template library to use as the source for this component's template. This option appears only if you select **Library** as your template type and **Other library** as the source of the library.
- **Library template name:** Name of a template that resides in the template library used by this component.

## Insert Anything into Report?

Yes, inserts a page footer

### Class

`rptgen.cform_page_footer`

### See Also

Page Header, PDF Page Layout



# Page Header

Insert a page header in a form-based report

## Description

Generates a page header in a Word or PDF report. A template associated with this component defines its fixed content and format and holes for filling the header with generated content. You can use this component to generate up to three types of headers per section: one for the first page, one for odd pages, and one for even pages.

You can define your report's page headers in templates. When you assign a template that defines headers to your report's Report Form or a Subform component, the Report Generator creates a Page Footer component for each page footer defined in each page layout defined by the template. It appends the page footer components that it creates for a particular template-defined page layout to the page layout component that it generates for that template-defined layout.

In addition to defining page headers in your templates, you can define them directly in the Report Explorer. For example, you can:

- Create page layout components in the Report Explorer and add header components to them.
- Add headers to layouts defined in your templates.
- Change the templates assigned to headers defined in your report templates.

The Report Generator generates a Template Hole component for each hole defined by a page header template. Every header has at least one hole component, a #start# component. You can define additional holes in the template that defines the header. You can add content to a header by appending components to the header holes in your report setup. The Report Generator generates header content by executing its hole components when it generates the parent page layout. The generated content applies to all pages of the specified header type. This means that you cannot use the Report Explorer to generate the header content for specific pages. You can, however, use Word and PDF fields, such as page number fields, in header templates to generate page numbers and other kinds of content that varies from page to page in a section.

To understand how layouts work, see "Define Page Layouts in a Form-Based Report Setup" on page 16-4.

## Page Header Options

The options indicate the type of page the header applies to and the template that defines the header's form.

- **Page type:** The type of page in this page layout that the header applies to.
  - **Default:** Header for odd pages of the section, even pages if you do not specify an even-page footer, and the first page if you do not specify a first-page footer.
  - **Even:** Header for even pages of the section.
  - **First:** Header for the first page of the section.
- **Template type:** Specify the template that defines the header content.

- **Library:** Select **Library** to select a template from a document part library. When you select this option, the **Source Library Options** appear.
- **File:** Select **File** to select a template file as the source of the header content.
- **Page Layout:** This option appears if the **Page Header** component was based on a header in the template assigned to a **Report Form** or **Subform** component in your report setup.
- **Source Library Options:** If you select **Library** as the template type, you can set these options.
  - **Report form library:** Template library of the template file assigned to this report setup's **Report Form** component.
  - **Parent subform library:** Template library used by the **Subform** that contains this **Header** component. This option appears only if this component is a descendant of a **Subform** component and the parent subform uses a library as the source of its template.
  - **Other library:** Template library of a specified template file.
- **Template:** If the template type is **File**, this option specifies the name of the template file that defines the header associated with this component. If the template type is **Library**, this option specifies the template file that contains the template library to use as the source for this component's template. This option appears only if you select **Library** as your template type and **Other library** as the source of the library.
- **Library template name:** Name of a template that resides in the template library used by this component.

## Insert Anything into Report?

Yes, inserts a page header

### Class

`rptgen.cform_page_header`

### See Also

Page Footer, PDF Page Layout, DOCX Page Layout, Subform

# Paragraph

Insert paragraph text into report

## Description

This component inserts a paragraph into the report. The paragraph text is taken from a child text component, or from text that you enter in the **Paragraph Text** field.

## Title Options

- **No paragraph title** (default): Specifies no title for the paragraph.
- **Get title from first child**: Gets the title of the paragraph from its first child component, which should be a Text component.
- **Custom title**: Specifies a custom title for the paragraph.

## Style Name

Specifies a style to use for the paragraph title for reports whose output format is one of the **from template** types, for example, **Direct PDF (from template)**. By default, this option specifies the **rgParagraphTitle** style defined by the Report Explorer's default document conversion templates. To modify the appearance of all paragraph titles in a report, customize the **rgParagraphTitle** style in the report template. To modify the appearance of this title only, specify another style defined in the report template. To specify another style:

- 1 In the Paragraph properties dialog box, set **Title Options** to one of these values:
  - **Get title from first child**
  - **Custom title**
- 2 Set **Style Name** to Specify.
- 3 In the **Style Name** text box, type a style name.

To use the custom style you specify, it must be a paragraph style (or a linked paragraph/character style for Word reports) defined in the template that you use to generate the report. For more information about template styles, see "Report Templates" on page 7-2.

In Microsoft Word, paragraph styles and text styles applied using a named style can interact when you apply both to the same text. This interaction occurs for formatting properties that paragraphs and text have in common, such as bold, italic, and underline.

Suppose that you apply the paragraph style **MyStyle** to a paragraph and that **MyStyle** specifies bold formatting. You then apply a text style named **BoldStyle** to text in the paragraph. The text formatted with **BoldStyle** toggles the paragraph style bold off. To ensure the text is bold you can create a text style that does not use bold but has the other properties you want to apply to text in that type of paragraph.

## Paragraph Text

Enter paragraph text into this field. If the **Paragraph** component has child components, the paragraph content is taken from the paragraph text and the child components. Otherwise, the

**Paragraph** component inserts text from this field. If the **Paragraph** component does not have any child components and you do not enter any text into this field, no text appears in the report.

To insert text to be computed when the report is generated, use the computed property notation, %<expr>, where *expr* is a MATLAB expression that evaluates to a string, .

## Style Name

Specifies a style to use with the paragraph text for reports whose output format is one of the `from template` types, for example, `Direct PDF (from template)`. By default, this option specifies the `rgParagraph` style defined by the Report Explorer's default document conversion templates. To modify the appearance of all paragraphs in a report, customize the `rgParagraph` style in the report template. To modify the appearance of this paragraph only, specify another style defined in the report template. To specify another style:

- 1 Set **Style Name** to Specify.
- 2 In the **Style Name** text box, type a style name.

To use the custom style you specify, it must be a paragraph style (or a linked paragraph/character style for Word reports) defined in the template that you use to generate the report. For more information about template styles, see "Report Templates" on page 7-2.

## Style

---

**Note** If you use the **Style Name** field to specify a style for the paragraph text, the style formats below override the corresponding formats specified in the style. For example, selecting **Bold** makes the text bold, even if the specified style specifies regular weight text.

---

- **Bold:** Makes the text bold.
- **Italic:** Makes the text italic.
- **Underline:** Underlines the text.
- **Strikethrough:** Strikes through the text.
- **Preserve white space:** Preserves sequences of white spaces in the text. If you select **Preserve white space**, the white space is preserved in all child components of that paragraph. If you do not select **Preserve white space**, multiple spaces are reduced to a single space. This option applies only to reports with an output type of (`from template`), such as `Direct PDF (from template)`. The option does not appear for other output types.

---

**Note** These format options override any corresponding formats defined in the custom style you specify. For example, selecting **Bold** makes the text bold, even if the specified style uses regular weight text.

---

To preserve white space in a PDF report, you must select this option at the paragraph level. PDF does not support preserving white space for selected **Text** children of a **Paragraph** component.

---

- **Color:** Specifies the color of the text.
  - Select a color from a list of colors.

- Enter a hexadecimal RGB value as #RRGGBB. For example, #0000ff is a shade of blue.
- Enter %<expr>, where expr is a MATLAB expression that evaluates to a color name or a hexadecimal RGB value.

## **Insert Anything into Report?**

Yes. Paragraph.

## **Class**

rptgen.cfr\_paragraph

## **See Also**

Chapter/Subsection, Empty Component, Image, Link, List, Table, Text, Title Page

## PDF Page Layout

Page layout in a PDF report

### Description

This component generates a page layout definition for a section of a PDF report. The page layout definition specifies the size and orientation of pages in the section, the sizes of the section's page margins, and the format and starting value of the section's page numbers. It can optionally define a watermark to appear in the background of each page in the section.

You can create instances of this component interactively or from report templates. For more information, see "Define Page Layouts in a Form-Based Report Setup" on page 16-4.

### Page Numbering

- **First page number:** Number of the first page in this page layout section.
  - **Auto:** The number of the first page of this layout. If you are using a template, the value is the first page number defined in the template from which this layout was generated. If you create this layout interactively, numbering continues from the previous page layout section.
  - **Specify:** Specify the first page number as an integer.
- **Page number format:** Format of the page numbers in this page layout section:
  - **None:** Use Arabic numerals, the default formatting for page numbers.
  - **Lower case alphabetic**
  - **Upper case alphabetic**
  - **Lower case roman numerals**
  - **Upper case roman numerals**
  - **Arabic numerals**
- **Section break:** Where to start this section:
  - **Next Page:** Start this page layout on a new page immediately following the last page of the previous section.
  - **Odd Page:** Start this page layout on a new page, immediately after the last page of the previous section. If the previous section ended on an odd page, insert an empty page at the end of the last section.
  - **Even Page:** Start this page layout on a new page, immediately after the last page of the previous section. If the previous section ended on an even page, insert an empty page at the end of the previous section.

### Page Margin Options

- **Page Margin:**
  - **Auto:** If this layout component was generated from a template, use the margin values specified by the template. Otherwise, use default values.

- **Specify:** Specify the size of the page margins in the form `valueUnits`. Use any of these values for units:
  - `cm` — centimeters
  - `in` — inches
  - `mm` — millimeters
  - `pc` — picas
  - `px` — pixels (default)
  - `pt` — points
- **Top:** Size of top page margin. The default value is `1in`.
- **Bottom:** Size of bottom page margin. The default value is `1in`.
- **Left:** Size of left page margin. The default value is `1in`.
- **Right:** Size of right page margin. The default value is `1in`.
- **Header:** Size of header area. The default value is `0.5in`.
- **Footer:** Size of footer area. The default value is `0.5in`.
- **Gutter:** Size of gutter (area for binding pages). The default value is `0px`.

## Page Size Options

- **Page Size:** If this layout component was generated from a template, selecting `Auto` uses the values specified in the template. Otherwise, selecting `Auto` uses default values.

Select `Specify` to enter your own page height, width, or orientation. Specify the units in the form `valueUnits`. Use any of these values for units:

- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `px` — pixels (default)
- `pt` — points

## Watermark Options

These options allow you to include a watermark in this section of your report. A watermark is an image that appears in the background of the page. For example, a watermark can indicate that the section is confidential or a draft. You can include a watermark in a page layout in your template. See “Watermarks in PDF Page Layouts” on page 13-145.

- **File name:** Specifies the path of the image file to use as the watermark for this section. If this layout component was generated from a template and the template specifies a watermark, displays the watermark image path specified in the template. You can override the value. If the template does not specify a path or you created this component interactively, no path name appears. In either case, you can specify an image path to create a watermark. The image file must use `.bmp`, `.jpg`, `.png`, `.svg`, `.ir`, or `.tiff` format.
- **Scaling:**

- **Auto:** Use the image size in the image file.
- **Specify:** Scale the image to the specified height and width.

## **Insert Anything into Report?**

Yes, inserts a page layout.

### **Class**

`rptgen.cform_pdf_page_layout`

### **See Also**

Page Footer, Page Header, DOCX Page Layout, `mlreportgen.dom.Watermark`



# Preformatted

Insert preformatted paragraph

## Description

This component inserts a preformatted paragraph into a report. A preformatted paragraph is a paragraph that uses line feeds to break its content into lines and sequences of spaces to indent the lines. For accurate indentation, the **Preformatted** component default style uses the `rgProgramListing` default style, which specifies a monospace font. You can insert a **Preformatted** component as a child of a **Chapter/Subsection** or **Table Entry** component.

---

**Note** This component applies only to template-based reports, such as **Direct PDF** (from **Template**). In XSL style sheet-based reports, for example, **web (HTML)**, it serves as a **Paragraph** component.

---

## Properties

- **Preformatted content to include in report:** Specifies the preformatted text to include in the report.
- **Style Name:** Specifies the style to use with the text. By default this option specifies `rgProgramListing`, a style defined in the default document conversion templates installed with the Report Explorer. To modify the appearance of all preformatted paragraphs in a report, customize the `rgProgramListing` style. To modify the appearance of this paragraph only, specify another style defined in the report template. To specify another style:

- 1 Set **Style Name** to `Specify`.
- 2 In the **Style Name** text box, type a style name.

To use the custom style you specify, it must be defined as a span style in the HTML or PDF template used with this report. For a Word template, it must be defined as a character or linked paragraph/character style. For more information about template styles, see “Report Templates” on page 7-2.

## Style

- **Bold:** Makes the text bold.
- **Italic:** Makes the text italic.
- **Show text as syntax-highlighted MATLAB code:** Uses colors to highlight MATLAB syntax, such as comments and variable names, in the text content of the component.
- **Color:** Specifies the color of the text.
  - Select a color from the list of colors.
  - Enter a hexadecimal RGB value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.
  - Enter `%<expr>`, where `expr` is a MATLAB expression that evaluates to a color name or a hexadecimal RGB value.

---

**Note** These format options override any corresponding formats defined in the custom style you specify. For example, selecting **Bold** makes the text bold, even if the specified style uses regular weight text.

---

## **Insert Anything into Report?**

Yes. Paragraph that uses white space to divide its content into lines and indent the lines.

For example,

```
This shows text
containing   multiple spaces
  an indented line,
and line feeds
in a Preformatted component.
```

### **Class**

rptgen.cfr\_preformatted

### **See Also**

Code, Text, Paragraph

# Stop Report Generation

Halt report generation

## Description

This component acts like **Stop** during report generation. You can use this component inside an `if/then` statement by using Logical and Flow Control components to halt the report-generation process when the specified condition is `true`. When report generation halts, an XML source file is created, but not converted.

## Confirmation Properties

- **Confirm before stopping generation:** Generates a confirmation dialog box before stopping report generation.
- **Confirmation question:** Specifies a confirmation question for the prompt. The default is “Stop generating the report?”
- **Halt button name:** Specifies a name for the button that stops report generation. The default is “Halt Generation”.
- **Continue button name:** Specifies a name for the button that continues report generation. The default is “Continue Generation”.

## Example

This example creates a simple report that takes a snapshot of the current figure. If there is no current figure, the report generation automatically halts:

```
[ - ] Report - figure-report.rpt
  [ - ] if (isempty(get(0,'CurrentFigure')))
    [ ] Stop Generation
  [ - ] Figure Loop - current
    [ - ] Chapter - <Title from SubComponent1>
      [ ] Figure Name
      [ ] Graphics Figure Snapshot
      [ ] Figure Prop Table - Figure Properties
```

## Insert Anything into Report?

No.

## Class

rptgen.crg\_halt\_gen

## See Also

Comment, Import File, Nest Setup File, Time/Date Stamp

## Subform

Create a subform

### Description

This component allows you to fill a hole in a form or subform with content based on a subform. Assigning a subform template to an instance of this component populates the instance with a set of hole and page layout components representing the subform template. During report generation, this component executes the **Template Hole** components in the order in which they appear in the subform template. The **Template Hole** components in turn execute their children to fill the holes in the output report form.

You can determine the generated content by appending appropriate content-generation components, for example, **Paragraph** components, to the **Template Hole** components that populate the Subform component. For more information, see “Create Multiform-Based Report Setups” on page 16-3.

### Subform Options

- **Template type:** Specify the template that defines the subform.
  - **Library:** Select **Library** to select a template from a document part library. When you select this option, the **Source Library Options** appear.
  - **File:** Select **File** to select a template file as the subform.
- **Source Library Options:** If you select **Library** as the template type, you can set these options.
  - **Report form library:** Template library of the Report Form that contains this component
  - **Parent subform library:** Template library used by the Subform component that contains this Subform component. This option appears only if this component is a child of another Subform component and the parent Subform component uses a library as the source of its template.
  - **Other library:** Template library of a specified template file.
- **Template:** If the template type is **File**, this option specifies the name of the template file that defines the subform associated with this component. If the template type is **Library**, this option specifies the template file that contains the template library to use as the source for this component's template. This option appears only if you select **Library** as your template type and **Other library** as the source of the library.
- **Library template name:** Select the name of the subform template from the library of the template in effect for this component.

### Insert Anything into Report?

Yes. Inserts a subform.

### Class

rptgen.cform\_subform

## **See Also**

PDF Page Layout, DOCX Page Layout, Template Hole

# Table

Insert parent of table

## Description

This component is a parent of a component hierarchy that you specify to insert a table into a report. Adding this component creates a hierarchy that defines a 2x2 table that you modify to define your specific table.

## Properties

- **Title:** Specifies a title for the table. Enter text or %<expr>. If you specify a table title, text in the form `Table #:` precedes the table title.
- **Title style name:** Specifies the style to use with the table title. To specify a style:
  - 1 Set the report's **File format** to one of the `from template` options, for example, `Direct PDF (from template)`.
  - 2 In the Table properties dialog box, set **Title style name** to `Specify`.
  - 3 In the **Title style name** text box, type a style name.

To take effect, the style you specify must be defined in a table style in the template that you use to generate the report. For more information about template styles, see "Report Templates" on page 7-2.
- **Number of columns:** Specifies the number of columns in the table. Enter a number or %<expr>. A table must have at least one column.
- **Table style name:** Specifies the style to use with the table. To specify a style:
  - 1 Set the report's **File format** to one of the `from template` options, for example, `Direct PDF (from template)`.
  - 2 Set **Table style name** to `Specify`.
  - 3 In the **Table style name** text box, type a style name.

To take effect, the style you specify must be a table style in the template that you use to generate the report. For more information about template styles, see "Report Templates" on page 7-2.
- **Table width options:** Determines the width of the table.
  - **Auto:** Sets the table width based on the table contents.
  - **Specify:** Enter the table width as a percentage of the page width (for example, 75%) or as an absolute width. When you specify an absolute width, you can include the units, for example, 5in. Supported units are inches (`in`), picas (`pc`), points (`pt`), and pixels (`px`). If you do not specify a unit, the default for template-based output types (for example, `Word from template`) is pixels. For all other output types, the default is points.
- **Table spans page width:** Spreads the table across the width of the page. If you clear this property, the table uses the **Table width options** setting.
- **Border:** Specifies whether to draw border lines around the outside edges of the table. For example, to draw a border line only at the top of the table, select `Top`.

- **Between columns:** Draw a vertical line on the right side of each column (except for the last column) in the table.

To override this setting for a specific column or table entry, use the **Column separator** property of the Table Column Specification or Table Entry components, respectively.

- **Between rows:** Draw a horizontal line at the bottom of each row (except for the last row) in the table.

To override this setting for a specific table column, row, or entry, use the **Row separator** property of the appropriate component: Table Column Specification, Table Row, or Table Entry.

- **Horizontal entry alignment:** Aligns the position of Table Entry component content relative to the left and right sides of a table column.
  - **Left:** Aligns content with the left side of the column
  - **Center:** Aligns content in the middle of the column
  - **Right:** Aligns content with the right side of the column
  - **Double justified:** Justifies the left and right sides of the entry content, to avoid ragged left and right alignment

To override this setting:

- For a specific table column, use the Table Column Specification **Entry horizontal alignment** property.
- For a specific table entry, use the Table Entry **Horizontal alignment** property.
- **Indent:** Specifies the amount by which to indent the table from the left edge of an HTML page or from the left margin of a Word page. The specified indent must be positive and may include an optional unit specifier. For example, you can specify `0.67in`. If you do not specify a unit, pixels is the assumed unit. Here are the unit abbreviations.
  - `in`
  - `cm`
  - `mm`
  - `pt`

---

**Note** To use this option, set the report's **File format** to one of the `from template` options, for example `Direct PDF (from template)`.

---

- **Rotate table 90 degrees:** For PDF and HTML output file formats, rotates the table 90 degrees counterclockwise to the direction of the text flow on the page.

## Insert Anything into Report?

Yes. Table.

### Class

`rptgen.cfr_ext_table`

## **See Also**

Table Body, Table Column Specification, Table Entry, Table Footer, Table Header, Table Row, Array-Based Table, Chapter/Subsection, Empty Component, Image, Link, List, Paragraph, Text, Title Page



# Table Body

Insert parent of table body

## Description

This component is a parent of the rows that define the body of a table.

This component must be a child of a Table component. Add Table Row components as children to define the content of the table body.

## Properties

- **Style Name:** Specifies the style to use with the table body. To specify a style:
  - 1 Set the report's **File format** to one of the from template options, for example, Direct PDF (from template).
  - 2 Set **Style Name** to Specify.
  - 3 In the **Style Name** text box, type a style name.

To take effect, the style you specify must be a table style defined in the template that you use to generate the report. For more information about template styles, see "Report Templates" on page 7-2.

- **Entry vertical alignment:** Positions table entry content relative to the top and bottom of the row in which the table entry appears.

To override this setting for a table header or footer, or for a table row within one of those table elements, use the **Entry vertical alignment** property for the Table Header, Table Footer, or Table Row component.

To override this setting for a specific table entry, use the Table Entry **Vertical alignment** property.

## Insert Anything into Report?

Yes. Table.

## Class

rptgen.cfr\_ext\_table\_body

## See Also

Table, Table Column Specification, Table Entry, Table Footer, Table Header, Table Row, Array-Based Table, Chapter/Subsection, Empty Component, Image, Link, List, Paragraph, Text, Title Page

## Table Column Specification

Specify table column properties

### Description

Specifies the format of a table column. Add a Table Column Specification component for only those columns that you do not want the default settings for the table.

### Properties

- **Column number:** Specifies a column number for the column to which this column specification applies. Enter a number or %<expr>. Avoid using the same column number for two column specifications in the same table.
- **Column name:** Specifies the name of this column. The name appears in the Outline pane of the Report Explorer. Enter text or a %<expr>.

A Table Entry component can use this name to specify that it starts or ends on this column.

- **Column width:** Specifies the width of the column.

To specify an absolute column width, specify a number or %<expr>. When you specify an absolute width, you can include the units, for example, 5in. Supported units are inches (in), picas (pc), points (pt), and pixels (px). If you do not specify a unit, the default for template-based output types (for example, Word from template) is pixels. For all other output types, the default is points.

You can use relative widths for columns. If you use relative widths for one column in a table, you must use relative widths for the other columns in the table. Specify 1\* for one column, as a baseline. For other columns, specify the width as a factor of the baseline column. The width of each column reflects its relative size. For example, suppose a two column table is 6 inches wide. The width of the first column is set to 1\*, and the width of the second column is set to 2\*. The width of the first column is 2 inches, and the width of the second column is 4 inches.

- **Entry horizontal alignment:** Justifies the position of table entries in the column, relative to the left and right sides of the column.

Use the **Horizontal entry alignment** setting of the Table component, or explicitly set this property:

- **Left:** Aligns content with the left side of the column.
- **Center:** Aligns content in the middle of the column.
- **Right:** Aligns content with the right side of the column.
- **Double justified:** Justifies the left and right sides the entry content, to avoid ragged left and ragged right alignment.

To override this setting for a specific table entry, use the Table Entry **Horizontal alignment property** for that table entry.

- **Column separator:** Use the **Between columns** setting of the Table component, or explicitly set the **Column separator** property.

- **True:** Draws a vertical line at the right edge of the column (except for the last column).
- **False:** Draws no vertical line at the right edge of the column.
- **Row separator:** Use the **Between rows** setting of the Table component, or explicitly set the **Row separator** property.
  - **True:** Draws a horizontal line at the bottom of each row in the column (except for the bottom row).
  - **False:** Does not draw a horizontal line at the bottom of each row in the column.

## Insert Anything into Report?

Yes. Table.

### Class

`rptgen.cfr_ext_table_colspec`

### See Also

Table, Table Body, Table Entry, Table Footer, Table Header, Table Row, Array-Based Table, Chapter/Subsection, Empty Component, Image, Link, List, Paragraph, Text, Title Page

## Table Entry

Insert table entry

### Description

Specifies the format of a table entry.

This component must be a child of a descendant of a **Table Row** component. Add **Paragraph**, **Image**, **List**, and other components to define the content of the table entry.

---

**Note** Spanning columns and rows is not supported when the **File format** for generating the report is set to Word Document (RTF) or Rich Text Format.

---

### Properties

- **Horizontal alignment:** Use the **Entry horizontal alignment** setting of the Table Column Specification component for the column in which the table entry appears, or explicitly set the **Horizontal alignment** property.
  - **Left:** Aligns content with the left side of the column.
  - **Center:** Aligns content in the middle of the column.
  - **Right:** Aligns content with the right side of the column.
  - **Double justified:** Justifies the left and right sides the entry content, to avoid ragged left and right alignment.
- **Vertical alignment:** Positions the table entry content relative to the top and bottom of the row in which the table entry appears.

Use this property to override the **Entry vertical alignment** setting of the Table Row component in which this table entry appears.

- **Column separator:** Use this property to override the **Column separator** setting of the Table Column Specification component for the column in which the table entry appears.
  - **True:** Draws a vertical line at the right edge of the column for this table entry.
  - **False:** Draws no vertical line at the right edge of the column for this table entry.
- **Row separator:** Use this property to override the **Row separator** setting of the Table Row component for the row in which the table entry appears.
  - **True:** Draws a horizontal line at the bottom of the row, below the table entry.
  - **False:** Does not draw a horizontal line at the bottom of the row, below the table entry.
- **Background color:** Specifies the background color of the table entry. You can:
  - Use Auto to apply the **Background Color** setting of the Table Row component in which the table entry appears.
  - Select a color from a list of colors.
  - Enter %<expr>.

- Enter an RGB (truecolor) value as #RRGGBB. For example, #0000ff is a shade of blue.
- **Span start column name:** Specifies the name of the column (as defined by the Table Column Specification component) to use as the first (left side) of a set of spanned columns for displaying the table entry content. This property becomes unavailable when the **File format** for generating the report is set to Word Document (RTF) or Rich Text Format.
- **Span end column name:** Specifies the name of the column (as defined by the Table Column Specification component) to use as the last (right side) of a set of spanned columns for displaying the table entry. This property becomes unavailable when the **File format** for generating the report is set to Word Document (RTF) or Rich Text Format.
- **Rows spanned:** Specifies the number of rows to span for the table entry. The spanning starts with the table row in which you define the table entry and extends below that row for the number of rows that you specify. This property becomes unavailable when the **File format** for generating the report is set to Word Document (RTF) or Rich Text Format.
- **Text orientation:** Rotates table entry text in 90 degree increments, relative to the page text flow. This property is available only if you select Acrobat PDF or Direct PDF (from template) as the output type file format.

To use the text orientation of the table row in which this table entry appears, select Auto.

To override the **Text orientation** setting for the Table Row component in which this table entry appears, select a rotation value.

- **Rotated text width:** Specifies the width of table entry text that you rotate (with the **Text orientation** property). This property is available only if you select Acrobat PDF or Direct PDF (from template) as the output type file format.

When you specify this width, you can include the units, for example, 5in. Supported units are inches (in), picas (pc), points (pt), and pixels (px). If you do not specify a unit, the default for template-based output types (for example, Word from template) is pixels. For all other output types, the default is points.

To avoid truncating the rotated text, set the **Rotated text width** to a value that allows the display of the longest line of text in the table row.

## Insert Anything into Report?

Yes. Table.

### Class

rptgen.cfr\_ext\_table\_entry

### See Also

Table, Table Body, Table Column Specification, Table Footer, Table Header, Table Row, Array-Based Table, Chapter/Subsection, Empty Component, Image, Link, List, Paragraph, Text, Title Page

## Table Footer

Insert parent of table footer

### Description

This component is a parent of the Table Row components that define a table footer.

### Properties

- **Style Name:** Specifies the style to use with the table footer. To specify a style:
  - 1 Set the report's **File format** to one of the from template options, for example, Direct PDF (from template).
  - 2 Set **Style Name** to Specify.
  - 3 In the **Style Name** text box, type a style name.

To take effect, the style you specify must be a table style defined in the template that you use to generate the report. For more information about template styles, see "Report Templates" on page 7-2.
- **Entry vertical alignment:** Positions the table entry content relative to the top and bottom of the table footer rows in which the table entries appear.

To override this setting for a specific row in the table footer, use the **Entry vertical alignment** property of the Table Row component for that row.

### Insert Anything into Report?

Yes. Table.

### Class

rptgen.cfr\_ext\_table\_foot

### See Also

Table, Table Body, Table Column Specification, Table Entry, Table Header, Table Row, Array-Based Table, Chapter/Subsection, Empty Component, Image, Link, List, Paragraph, Text, Title Page

# Table Header

Insert parent of table header

## Description

This component is a parent of the Table Row components that define a table header.

## Properties

- **Style Name:** Specifies the style to use with the table header. To specify a style:
  - 1 Set the report's **File format** to one of the from template options, for example, Direct PDF (from template).
  - 2 Set **Style Name** to Specify.
  - 3 In the **Style Name** text box, type a style name.

To take effect, the style you specify must be a table style defined in the template that you use to generate the report. For more information about template styles, see "Report Templates" on page 7-2.
- **Entry vertical alignment:** Positions the table entry content relative to the top and bottom of the table header rows in which the table entries appear.

To override this setting for a specific row in the table header, use the **Entry vertical alignment** property of the Table Row component for that row.

## Insert Anything into Report?

Yes. Table.

## Class

rptgen.cfr\_ext\_table\_head

## See Also

Table, Table Body, Table Column Specification, Table Entry, Table Footer, Table Row, Array-Based Table, Chapter/Subsection, Empty Component, Image, Link, List, Paragraph, Text, Title Page

## Table Row

Insert parent of table row entries

### Description

This component is a parent of Table Entry components that define a table row.

### Properties

- **Entry vertical alignment:** Positions the table entry content relative to the top and bottom of the table row in which the table entries appear.

Use this property to override the **Entry vertical alignment** setting of the Table Header, Table Footer, or Table Body component in which the table row appears.

- **Row separator:** Use this property to override the **Row separator** setting of the Table component.
  - **True:** Draws a horizontal line at the bottom of the row (except for the last row).
  - **False:** Does not draw a horizontal line at the bottom of the row.
- **Background color:** Specifies the background color of the table row. You can:
  - Use **Auto** for the background color that the report style sheet specifies, which for style sheets provided with MATLAB Report Generator is white by default.
  - Select a color from a list of colors.
  - Enter %<expr>.
  - Enter an RGB (truecolor) value as #RRGGBB. For example, #0000ff is a shade of blue.
- **Row height:** Specifies the height of the table row.

To let the table contents automatically set the row height, use **Auto**.

To specify an absolute height for this table row, select **Specify** and enter the height in inches (**in**), picas (**pc**), or points (**pt**).

- **Text orientation:** Rotates text in the table entries in this table row, relative to the page text flow. This property is available only if you select **Acrobat PDF** or **Direct PDF (from template)** as the output type file format.

To override the text rotation for a specific table entry, use the Table Entry **Text orientation** property for that table entry.

- **Rotated text width:** Specifies the width of table entry text that you rotate with the **Text orientation** property. This property is available only if you select **Acrobat PDF** or **Direct PDF (from template)** as the output type file format.

When you specify this width, you can include the units, for example, **5in**. Supported units are inches (**in**), picas (**pc**), points (**pt**), and pixels (**px**). If you do not specify a unit, the default for template-based output types (for example, **Word from template**) is pixels. For all other output types, the default is points.

To avoid truncating the rotated text, set the **Rotated text width** to a value that allows the display of the longest line of text in the table row.



To override the rotated text width for a specific table entry in the table row, use the Table Entry **Rotated text width** property.

## **Insert Anything into Report?**

Yes. Table.

### **Class**

rptgen.cfr\_ext\_table\_row

### **See Also**

Table, Table Body, Table Column Specification, Table Entry, Table Footer, Table Header, Array-Based Table, Chapter/Subsection, Empty Component, Image, Link, List, Paragraph, Text, Title Page

## Template Hole

Fills a hole in a forms-based report

### Description

This component fills a hole in a form-based report defined by one of the report's templates. When you assign a template to a report's Report Form component or to one of its Subform components, the Report Generator creates one of these components for each hole defined in the template and appends it to the form or subform component as a child. At report generation time, the form or subform component executes each of its hole components. Each hole component executes its children. You can thus fill the holes in a report by appending **Paragraph** and other content-generation components to the holes.

The Report Generator creates hole components for each hole that you define explicitly in a template. The hole component's **Template Hole** identifier property is set to the identifier that you assign to the hole in the template. This identifier allows you to determine which hole defined in a template the **Template Hole** component fills.

In addition to hole components based on holes you defined, for Word and PDF templates, the Report Generator creates a hole, called a section hole, for each page layout section defined by the template. It assigns the identifier **#start#** to the first section hole and the identifiers **#sect2#**, **#sect3#**, and so on, to subsequent section holes. You can append content to both types of holes.

To define holes, see:

- "Add Holes in a Microsoft Word Template" on page 13-120
- "Add Holes in HTML and PDF Templates" on page 13-132

For an example that shows how to define holes in a template and how they appear in a form-based report, see "Create a Simple Form-Based Setup" on page 16-6.

### Properties

Display properties of holes.

- **Hole identifier:** Identifier of the hole.
- **Hole type:** The hole type associated with the hole identifier:
  - An inline hole is for document elements that a paragraph can contain: Text, Image, Link.
  - A block hole can contain the same kinds of document elements as an inline hole, plus block type of content such as paragraphs, tables, lists, subforms, images, and snapshots.
- **Hole description:** The hole description from the template.
- **Default style name:** If the template that defines this hole specifies a default style name to apply to text that fills this hole, this field displays the default name. To use the default name with a **Paragraph** or **Text** component appended to this hole, select **Auto** as the value of the **Paragraph** or **Text** component's **Style name** property. To override the default style name, select the **Style name** property's **Specify** option. If the template does not define a default style name for this hole's content, the Report Generator uses Paragraph as a style name for Paragraph content and omits a style name for Text content.

## **Insert Anything into Report?**

Content generated by this hole's children

### **Class**

rptgen.cform\_template\_hole

### **See Also**

Subform

## Text

Format and insert text into report

### Description

This component formats and inserts text into the report. For Word and PDF reports, it must have the **Paragraph** component as its parent.

### Properties

- **Text to include in report:** Specifies text to include in the report. The specified text can include computed property expressions of the form `%<expr>` where `expr` is a MATLAB expression that evaluates to a string when the report is generated. For example, the text “This report was generated on %<date>.” inserts the specified text in the report with %<date> replaced by the date on which the report is generated.
- **Style Name:** Specifies the style to use with the text for reports whose output format is one of the from template types, for example, `Direct PDF (from template)`. By default, this component's content inherits the style of the paragraph in which it is inserted. To override the paragraph style, specify another style defined in the report template. To specify another style:

- 1 Set **Style Name** to Specify.
- 2 In the **Style Name** text box, type a style name.

The custom style you specify must be a span style for HTML and PDF reports or a character or linked paragraph/character style for Word reports defined in the template that you use to generate the report. For more information about template styles, see “Report Templates” on page 7-2.

In Microsoft Word, paragraph styles and text styles applied using a named style can interact when you apply both to the same text. This interaction occurs for formatting properties that paragraphs and text have in common, such as bold, italic, and underline.

For example, you apply the paragraph style `MyStyle` to a paragraph and that `MyStyle` specifies bold formatting. Then, you apply a text style named `BoldStyle` to text in the paragraph. The text formatted with `BoldStyle` toggles the paragraph style bold off for that text. To ensure the text is bold, create a text style that does not use bold but has the other properties you want to apply to text in that type of paragraph.

### Style

- **Bold:** Makes the text bold.
- **Italic:** Makes the text italic.
- **Underline:** Underlines the text.
- **Strikethrough:** Strikes through the text.
- **Subscript:** Formats text as a subscript, in a smaller font than the other text, set slightly below the other text.

- **Superscript:** Formats text as a superscript, in a smaller font than the other text, set slightly above the other text.
- **Preserve white space:** Preserves sequences of spaces in the text content of this component. Deselecting this option causes sequences of spaces to be reduced to one space. This option does not appear if this component is the child of a **Paragraph** component that has its **Preserve white space** option selected. The **Preserve white space** option applies to all children of that **Paragraph** component. This option applies only to HTML (from template), Single-File HTML (from template, or Word (from template) reports. The **Preserve white space** option does not appear for other output types.
- **Color:** Specifies the color of the text.
  - Select a color from the list of colors.
  - Enter a hexadecimal RGB value as #RRGGBB. For example, #0000ff is a shade of blue.
  - Enter %<expr>, where expr is a MATLAB expression that evaluates to a color name or a hexadecimal RGB value.

---

**Note** These format options override any corresponding formats defined in the custom style you specify. For example, selecting **Bold** makes the text bold, even if the specified style uses regular weight text.

---

## Insert Anything into Report?

Yes. Text.

### Class

rptgen.cfr\_text

### See Also

Chapter/Subsection, Empty Component, Image, Link, List, Paragraph, Table, Title Page

## Time/Date Stamp

Insert time and date of report generation into report

### Description

This component inserts the time and date of the report generation into your report as text. It must have the **Paragraph** or **Chapter/Subsection** component as its parent.

### Prefix

**Include text before stamp** : Includes text before the time/date stamp. Specify the text in the corresponding field.

### Time Stamp Properties

- **Include current time in stamp**: Inserts the current time into the time/date stamp.
- **Time display**: Specifies the appearance of the time display. Options include:
  - 12-hour
  - 24-hour
- **Time Separator**: Specifies a separation marker between hours, minutes, and seconds. Options include:
  - Blank space ( ): Formats time as Hour Minute Second
  - Colon (:): Formats time as Hour:Minute:Second
  - Period (.): Formats time as Hour.Minute.Second
  - None ( ) : Formats time as HourMinuteSecond
- **Include seconds in time stamp**: Displays seconds in the time/date stamp.

### Date Stamp Properties

- **Include current date in stamp**: Inserts the current date in the time/date stamp.
- **Date order**: Specifies the order in which the day, month, and year appear. Options include:
  - Day Month Year
  - Month Day Year
  - Year Month Day
- **Date separator**: Specifies a separation marker between day, month, and year. Options include:
  - Blank space ( ): Displays date as Day Month Year
  - Colon (:): Displays date as Day:Month:Year
  - Slash (/): Displays date as Day/Month/Year
  - Period (.): Displays date as Day.Month.Year
  - None ( ): Displays date as DayMonthYear

- **Month display:** Specifies how the month displays. Options include:
  - Long (December)
  - Short (Dec)
  - Numeric (12)
- **Year display:** Specifies how the year displays. Options include:
  - Long (2007)
  - Short (07)

## Preview

This pane displays the time/date stamp to appear in the report.

## Insert Anything into Report?

Yes. Text.

## Class

rptgen.crg\_tds

## See Also

Comment, Import File, Nest Setup File, Stop Report Generation

# Title Page

Insert title page at beginning of report

## Description

This component inserts a title page at the beginning of the report. To use the **Title Page** component, you need to have at least one **Chapter** component in your report. The **Title Page** component must be a child of the top-level **Report** or **Report Form** component.

For PDF and HTML reports, you can use the Style Sheet Editor to position title page elements (for example, title, copyright, and images) anywhere on the front or reverse side of the title page in any order. You can specify the size, color, weight, and slant of text elements. For details, see “Modify Title Page Properties” on page 9-14.

## Properties

The text fields on this property pane support the %<VariableName> notation.

## Main Tab

### Title

- **Title:** Specifies the title of the report. The title is in a large font.
- **Subtitle:** Specifies the subtitle of the report. The subtitle is in a smaller font under the title.

### Options

- **Author:**
  - **Custom(default):** Specifies the author of the report.
  - **No author:** Does not specify an author name.
  - **Automatic author:** Automatically includes your user name as the author name.

The author name appears under the subtitle, in a smaller font than the subtitle.

- **Include report creation date:** Includes the date that the report is created. Choose the date format in the corresponding list.
- **Include copyright holder and year:** Includes copyright holder and year information.
- **Display legal notice on title page:** Includes the legal notice, report creation date, and copyright information on the title page of PDF and Microsoft Word reports.

## Image Tab

### File

- **File name:** Specifies the file name of an image to appear under the subtitle, on the title page.
- **Copy to local report files directory:** Copies the image file into the folder in which the report file is located.



## Display Options

- **Scaling:** Controls size of the image, as displayed in a browser. Making an image larger using this option does not affect the storage size of the image, but the quality of the displayed image may decrease as you increase or decrease the size of the displayed image.

Generally, to achieve the best and most predictable display results, use the default setting of `Use image size`.

- `Use image size`: Causes the image to appear the same size in the report as on screen (default).
- `Fixed size`: Specifies the number and type of units.
- `Zoom`: Specifies the percentage, maximum size, and units of measure.
- **Size**: Specifies the size of the snapshot in the form `w h` (width, height). This field is active only if you choose `Fixed size` in the **Scaling** list
- **Alignment**: Only reports in PDF or RTF format support this property. Options include:
  - Auto
  - Right
  - Left
  - Center

## Abstract Tab

- **Abstract Text**: Specifies an optional abstract for the report.
- **Style Name**: Specifies the style to use with the abstract text. To specify a style:
  - 1 Set the report's **File format** to one of the `from template` options, for example, `Direct PDF (from template)`.
  - 2 Set **Style Name** to `Specify`.
  - 3 In the **Style Name** text box, type a style name.

To take effect, the style you specify must be a paragraph (or a linked paragraph/character style for Word reports) defined in the template that you use to generate the report. For example, if you use a Word template that defines a `Normal` style, you can enter `Normal` as the style name. For more information about template styles, see "Report Templates" on page 7-2.

## Style

---

**Note** If you use the **Style Name** field to specify a style for this text, the style formats below override the corresponding formats specified in the style. For example, selecting **Bold** makes the text bold, even if the specified style specifies regular weight text.

---

- **Bold**: Makes the text bold.
- **Italic**: Makes the text italic.
- **Underline**: Underlines the text.
- **Strikethrough**: Strikes through the text.

- **Retain spaces and carriage returns:** Formats the text in the generated report as you entered it.
- **Show text as syntax-highlighted MATLAB code:** Shows the text as syntax-highlighted MATLAB code.
- **Color:** Specifies the color of the text.

## Legal Notice Tab

- **Legal Notice Text:** Specifies an optional legal notice for the report.
- **Style Name:** Specifies the style to use with the legal notice text. To specify a style:
  - 1 Set the report's **File format** to one of the from template options, for example, Direct PDF (from template).
  - 2 Set **Style Name** to Specify.
  - 3 In the **Style Name** text box, type a style name.

To take effect, the style you specify must be a paragraph (or a linked paragraph/character style for Word reports) defined in the template that you use to generate the report. For example, if you use a Word template that defines a Normal style, you can enter Normal as the style name. For more information about template styles, see "Report Templates" on page 7-2.

## Style

---

**Note** If you use the **Style Name** field to specify a style for this text, the style formats below override the corresponding formats specified in the style. For example, selecting **Bold** makes the text bold, even if the specified style specifies regular weight text.

---

- **Bold:** Makes the text bold.
- **Italic:** Makes the text italic.
- **Underline:** Underlines the text.
- **Strikethrough:** Strikes through the text.
- **Retain spaces and carriage returns:** Formats the text in the generated report as you entered it.
- **Show text as syntax-highlighted MATLAB code:** Shows the text as syntax-highlighted MATLAB code.
- **Color:** Specifies the color of the text.

## Insert Anything into Report?

Yes. Title page.

## Class

rptgen.cfr\_titlepage

## See Also

Chapter/Subsection, Empty Component, Image, Link, List, Paragraph, Table, Text

# Variable Table

Insert table that displays all the variables in the MATLAB workspace

## Description

This component inserts a table that displays all the variables in the MATLAB workspace.

---

**Tip** Find all the variables in the MATLAB workspace by typing `whos` at the command line.

---

## Source Workspace

### Read variables from:

- `Base workspace`: Reads variables from the MATLAB workspace.
- `MAT-file`: Reads variables from a binary file with a `.mat` extension. Use the `%<VariableName>` notation.

## Table Title

- **Table title:**
  - `Automatic (Variables from MATLAB workspace)`: Sets the table title to the name of a MATLAB variable.
  - `Custom`: Specifies a custom title.
- **Table Columns:**
  - `Variable dimensions (MxN)`: Includes the size of the variable.
  - `Variable memory bytes`: Includes the number of bytes of memory consumed by the variable.
  - `Variable class`: Includes the variable class.
  - `Variable value`: Includes the value of the variable.

---

**Note** Large variable arrays collapse to `[MxN CLASS]`. For example, if you have a 300-by-200 double array, it appears in the report as `[300x200 DOUBLE]`.

---

## Example

The following is an example of a variable table that includes size, memory bytes, and value information in the table columns.

Name	Size	Bytes	Value
aCell	1x2	238	{ [ 1 2 3 4 ] Speed (kph) }
aNumber	1x1	8	1
aString	1x11	22	Speed (kph)

<b>Name</b>	<b>Size</b>	<b>Bytes</b>	<b>Value</b>
aStructure	1x1	302	[struct w/ fields. Inputs, Outputs]
aVector	1x4	32	[ 1 2 3 4 ]

### **Insert Anything into Report?**

Yes. Table.

### **Class**

rptgen.cml\_whos

### **See Also**

Evaluate MATLAB Expression, Insert Variable, MATLAB Property Table, MATLAB Toolbox Version Number

# While Loop

Iteratively execute child components while a specified condition is true

## Description

This component iteratively executes its child components while a specified condition is true. The `While Loop` component must have at least one child component; the purpose of this component is to run its children several times. If it does not have any children, this component does not add anything to the report.

---

**Tip** Limit the number of repetitions to prevent infinite loops.

---

## Logic Properties

- **Continue looping if this expression is true:** Specifies an expression to evaluate. This expression must be a valid MATLAB expression that evaluates to 1 or 0 (true or false).

For example, if  $a = 1$ ,  $b = 2$ , and  $c = 3$ , the following command:

```
d=(a>b/c)
```

returns:

```
d = 1
```

Because 1 is greater than  $b/c$  ( $2/3$ ), this expression is true and evaluates to 1.

- **Limit number of loops to:** Allows you to prevent infinite loops. Use the left and right arrows to increase or decrease the number of loops.
- **Initialize with this expression:** Initializes the loop with a valid MATLAB expression.

## Insert Anything into Report?

Yes, if it has a child component.

## Class

```
rptgen_lo.clo_while
```

## See Also

For Loop, Logical Else, Logical Elseif, Logical If, Logical Then

# Report Explorer

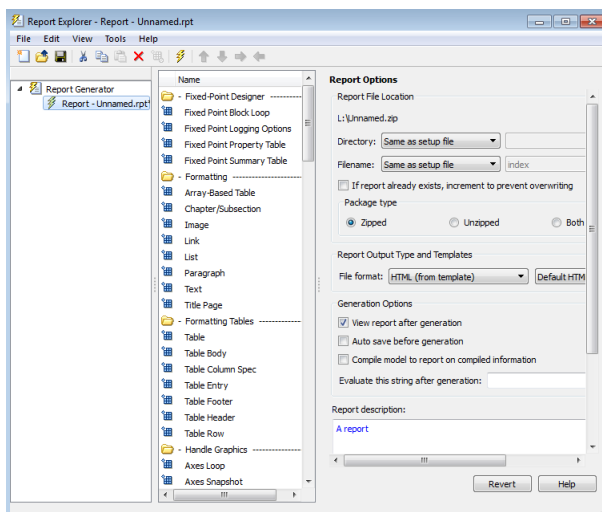
Design and generate reports on MATLAB applications

## Description

The **Report Explorer** app enables you to design and generate reports interactively.

Use the Report Explorer to to:

- Create and modify report setup files.
- Apply templates or style sheets to format the generated report.
- Specify the report file format.
- Generate reports.



## Open the Report Explorer App

- MATLAB Toolstrip: On the **Apps** tab, under **Database Connectivity and Reporting**, click the **Report Generator** app icon.
- MATLAB command prompt: Enter report.

## Examples

- “Create a Report Setup File” on page 2-10
- “Add Report Content Using Components” on page 2-11
- “Generate a Report” on page 2-31

## See Also

### Functions

report | rptlist | setedit | rptconvert

### Topics

“Create a Report Setup File” on page 2-10

“Add Report Content Using Components” on page 2-11

“Generate a Report” on page 2-31

“Report Setup”

“Work with Components”

“Format Reports”

“Generate Reports”

“Manage Report Conversion Templates”

“Customize Report Conversion Templates”

### Introduced before R2006a





# Functions

---

## **compwiz**

Create custom MATLAB Report Generator components

### **Syntax**

```
compwiz  
compwiz ('-browse')  
compwiz (rptgen.cfr_list)
```

### **Description**

The Create Component dialog box creates a framework for custom report components. For more information, see “Create Custom Components” on page 8-2.

- `compwiz` with no arguments displays the Component Editor in the Report Explorer.
- `compwiz ('-browse')` displays a list of components from which to derive a new component.
- `compwiz (rptgen.cfr_list)` initializes the Component Editor with the settings of the referenced components.

### **See Also**

`setedit` | `report` | `rptconvert` | `rptlist`

### **Topics**

“Create Custom Components” on page 8-2

# docview

View or perform operations on Word document

## Syntax

```
docview(filename)
[status,message] = docview(filename)
docview(filename,operation1,...,operationN)
[status,message] = docview(filename,operation1,...,operationN)
```

## Description

`docview(filename)` opens and displays the specified file in Microsoft Word or an editor compatible with Word, such as OpenOffice. Word or an editor compatible with Word must be installed.

`[status,message] = docview(filename)` also returns success or failure and, if appropriate, an error message.

`docview(filename,operation1,...,operationN)` specifies one or more operations to perform in Word. The operations require Microsoft Word on a Windows platform.

`[status,message] = docview(filename,operation1,...,operationN)` also returns success or failure and, if appropriate, an error message.

## Examples

### Open a Report in Word

Open a document in RTF format in Word. This example assumes the document `magic-squares.rtf` is on the MATLAB path.

```
docview('magic-square.rtf');
```

### Print a Document

Print a document. This example assumes the document `mydoc.doc` is on the MATLAB path.

```
docview('mydoc.doc','printdoc');
```

### Use Several docview Options

Use `docview` to update TOC fields in a document, convert the document to PDF, and close a Word document. This command creates the document `mytocdoc.pdf` in the current folder. This example assumes the document `mytocdoc.docx` is on the MATLAB path.

```
docview('mytocdoc', 'updatedocxfields', 'convertdocxtopdf', 'closeapp');
```

## Input Arguments

### **filename** — Name of file to view, print, or update

character vector

Name of file to view, print, or update, specified as the full path name.

### **operation1, ..., operationN** — Operations to perform in Word

'updatefields' | 'updatedocxfields' | 'convertdocxtopdf' | ...

Operations to perform in Microsoft Word on a Windows platform, specified as one or more of these values:

- 'updatefields' — Update fields, such as the TOC.
- 'updatedocxfields' — Update fields in a DOCX document.
- 'convertdocxtopdf' — Convert a DOCX document to PDF.
- 'showdocxaspdf' — Convert a DOCX document to PDF and open in a PDF viewer.
- 'unlinkdocxsubdoc' — Replace links to other documents with the content of those documents. To learn about subdocuments, see `mlreportgen.dom.DOCXSubDoc`.
- 'printdoc' — Print the document. The document must be open in Word before you use this option.
- 'printdocscaled' — Prints the document scaled to the locale-specific page size, such as US Letter or A4.
- 'savedoc' — Save the document.
- 'closedoc' — Close the document.
- 'closeapp' — Close Word, if no document is open.

## Output Arguments

### **status** — Success status

0 | 1

Success status of the specified actions, returned as 0 when the action was not completed and 1 for success.

### **message** — Error or warning information

character vector

Error or warning information, returned as a character vector.

## See Also

`rptview` | `mlreportgen.dom.DOCXSubDoc` | `mlreportgen.utils.word` | `mlreportgen.utils.WordDoc`

**Introduced before R2006a**

# append

**Class:** `mlreportgen.dom.Container`

**Package:** `mlreportgen.dom`

Append DOM object to container

## Syntax

```
domObjOut = append(containerObj, domObj)
```

## Description

`domObjOut = append(containerObj, domObj)` appends the DOM object to the specified container object.

## Input Arguments

**containerObj** — Container object to append DOM object to

`mlreportgen.dom.Container` object

Container object to append DOM object to, specified as an `mlreportgen.dom.Container` object.

**domObj** — DOM document element object to append

DOM object

DOM document element object to append, specified as a DOM object.

- `mlreportgen.dom.CustomElement`
- `mlreportgen.dom.Container`
- `mlreportgen.dom.DocumentPart`
- `mlreportgen.dom.FormalTable`
- `mlreportgen.dom.Group`
- `mlreportgen.dom.ExternalLink`
- `mlreportgen.dom.HTML`
- `mlreportgen.dom.HTMLFile`
- `mlreportgen.dom.Image`
- `mlreportgen.dom.InternalLink`
- `mlreportgen.dom.LinkTarget`
- `mlreportgen.dom.OrderedList`
- `mlreportgen.dom.Paragraph`
- `mlreportgen.dom.RawText`
- `mlreportgen.dom.Table`
- `mlreportgen.dom.Text`

- `mlreportgen.dom.TemplateHole`
- `mlreportgen.dom.UnorderedList`

## Output Arguments

### **domObjOut** — Appended document element

document element object

Appended document element, returned by one of these DOM objects:

- `mlreportgen.dom.CustomElement`
- `mlreportgen.dom.Container`
- `mlreportgen.dom.DocumentPart`
- `mlreportgen.dom.FormalTable`
- `mlreportgen.dom.Group`
- `mlreportgen.dom.ExternalLink`
- `mlreportgen.dom.HTML`
- `mlreportgen.dom.HTMLFile`
- `mlreportgen.dom.Image`
- `mlreportgen.dom.InternalLink`
- `mlreportgen.dom.LinkTarget`
- `mlreportgen.dom.OrderedList`
- `mlreportgen.dom.Paragraph`
- `mlreportgen.dom.RawText`
- `mlreportgen.dom.Table`
- `mlreportgen.dom.Text`
- `mlreportgen.dom.TemplateHole`
- `mlreportgen.dom.UnorderedList`

## Examples

### **Append Content to Container**

Create a container object.

```
import mlreportgen.dom.*;
rpt = Document('MyReport', 'docx');

c = Container();
```

Append content to the container and append the container to the report.

```
append(c, Paragraph('Hello'));
append(c, Table(magic(5)));
append(rpt, c);
```

Close and generate the report.

```
close(rpt);  
rptview(rpt.OutputPath);
```

**See Also**

[mlreportgen.dom.Container](#) | [mlreportgen.dom.Group](#)

**Topics**

“Add Content as a Group” on page 13-12

**Introduced in R2015a**

## clone

**Class:** `mlreportgen.dom.Container`

**Package:** `mlreportgen.dom`

Copy container object

### Syntax

```
clonedContainer = clone(sourceContainer)
```

### Description

`clonedContainer = clone(sourceContainer)` copies (clones) the specified container.

### Input Arguments

**sourceContainer** — Container object to copy

`mlreportgen.dom.Container` object

Container object to copy, specified as an `mlreportgen.dom.Container` object.

### Output Arguments

**clonedContainer** — Copied container object

`mlreportgen.dom.Container` object

Copied container object, returned as an `mlreportgen.dom.Container` object.

## Examples

### Copy Container Object

Create a container object. Microsoft Word output ignores the HTML container element tag (in this example, the `div` tag).

```
import mlreportgen.dom.*;  
rpt = Document('MyReport', 'docx');
```

```
c = Container();
```

Color all of the text in this container red.

```
c.Style = {Color('red')};
```

Append content to the container and append the container to the report.

```
append(c, Paragraph('Hello'));  
append(c, Table(magic(5)));  
append(rpt, c);
```



Clone the container.

```
clonedC = clone(c);
```

Append the cloned container to the report.

```
append(rpt, clonedC);
```

Close and generate the report.

```
close(rpt);  
rptview(rpt.OutputPath);
```

## **See Also**

`mlreportgen.dom.Container` | `mlreportgen.dom.Group`

## **Topics**

“Add Content as a Group” on page 13-12

## **Introduced in R2015a**

## append

**Class:** `mlreportgen.dom.CustomElement`

**Package:** `mlreportgen.dom`

Append HTML content to custom element

### Syntax

```
domObjOut = append(customElementObj, domObj)
```

### Description

`domObjOut = append(customElementObj, domObj)` appends an element to a custom element.

### Examples

#### Append Text to a Custom Element

This example shows how to add a custom element that provides a check box in an HTML report.

Create and a custom element and append text to it.

```
import mlreportgen.dom.*;
d = Document('test');

input1 = CustomElement('input');
input1.CustomAttributes = {
    CustomAttribute('type', 'checkbox'), ...
    CustomAttribute('name', 'vehicle'), ...
    CustomAttribute('value', 'Bike'), ...
};
append(input1, Text('I have a bike'));
```

Append the custom element to an ordered list and display the report.

```
ol = OrderedList({input1});
append(d, ol);

close(d);
rptview('test', 'html');
```

### Input Arguments

**customElementObj** — Custom element to append content to

`mlreportgen.dom.CustomElement` object

Custom element to append content to, specified as an `mlreportgen.dom.CustomElement` object.

**domObj** — DOM object to append to custom element

DOM object

DOM object to append to the custom element.

## **Output Arguments**

### **domObjOut — DOM object appended to custom element**

DOM object

DOM object appended to a custom element, represented by a DOM object.

## **See Also**

`mlreportgen.dom.CustomElement` | `mlreportgen.dom.CustomAttribute`

## **Topics**

“Add Content to a Report” on page 13-10

## **Introduced in R2014b**

## addHTML

**Class:** mlreportgen.dom.Document

**Package:** mlreportgen.dom

Append HTML string to document

### Syntax

```
htmlObjOut = addHTML(documentObj,htmlText)
```

### Description

htmlObjOut = addHTML(documentObj,htmlText) converts a string of HTML text to a group of DOM objects and appends the group to the Document object documentObj.

### Input Arguments

**documentObj** — Document to append content to

mlreportgen.dom.Document object

Document object to append content to, specified as an mlreportgen.dom.Document object.

**htmlText** — HTML text

character vector

HTML text, specified as a character vector.

Example: '`<p><b>Hello</b> <i style="color:green">World</i></p>`'

### Output Arguments

**htmlObjOut** — HTML object with appended content

mlreportgen.dom.HTML object

HTML object with appended content, returned as an mlreportgen.dom.HTML object.

### Examples

#### Append HTML Text to Document

Create an HTML object from HTML text to use for a Microsoft Word report.

```
import mlreportgen.dom.*;
rpt = Document('HTMLToWordReport','docx');
htmlObj = addHTML(rpt,...
    '<p><b>Hello</b> <i style="color:green">World</i></p>');
```

Generate the Word report.

```
close(rpt);  
rptview(rpt.OutputPath);
```

## Tips

- MATLAB Report Generator `mlreportgen.dom.HTML` or `mlreportgen.dom.HTMLFile` objects typically cannot accept the raw HTML output of third-party applications, such as Microsoft Word, that export native documents as HTML markup. In these cases, your Report API report generation program can use the `mlreportgen.utils.html2dom.prepHTMLString` and `mlreportgen.utils.html2dom.prepHTMLFile` functions to prepare the raw HTML for use with the `mlreportgen.dom.HTML` or `mlreportgen.dom.HTMLFile` objects. Typically, your program will have to further process the prepared HTML to remove valid but undesirable objects, such as line feeds that were in the raw content.
- By default, the DOM API uses a base font size of 12 points to convert `em` units to actual font sizes. For example, a font size specified as `2em` converts to 24 points. To specify a different base font size, add your content to a report by using an `mlreportgen.dom.HTML` object. Set the `EMBaseFontSize` property of the object to the base font size. For example, if you set the `EMBaseFontSize` property to 14, a font size of `2em` converts to 28 points.

## See Also

`mlreportgen.dom.HTML` | `mlreportgen.dom.HTMLFile` |  
`mlreportgen.utils.html2dom.prepHTMLFile` |  
`mlreportgen.utils.html2dom.prepHTMLString`

## Topics

“Convert HTML Content to a DOM Object” on page 13-105

“Prepare HTML for Conversion to a DOM Object” on page 13-108

“Requirements for Converting HTML to a DOM Object” on page 13-110

## Introduced in R2015a

## addHTMLFile

**Class:** mlreportgen.dom.Document

**Package:** mlreportgen.dom

Append HTML file contents to document

### Syntax

```
htmlObjOut = addHTMLFile(documentObj,htmlFilePath)
```

### Description

`htmlObjOut = addHTMLFile(documentObj,htmlFilePath)` appends HTML file contents to a document.

### Input Arguments

**documentObj** — Document to append content to

mlreportgen.dom.Document object

Document object to append content to, specified as an mlreportgen.dom.Document object.

**htmlFilePath** — HTML file path

character vector

HTML file path, specified as a character vector.

### Output Arguments

**htmlObjOut** — HTML object with appended content

mlreportgen.dom.HTMLFile object

HTML object with appended content, returned as an mlreportgen.dom.HTMLFile object.

### Examples

#### Append HTML File Contents to a Document

In a text editor, create a file and enter this text:

```
<!DOCTYPE html>
<html>

<head>
  <title>My First HTML</title>
</head>

<body>
```

```
<p>This is the <b>first</b> paragraph.</p>
<p>This is the <b>second</b> paragraph</p>
</body>
</html>
```

Save the file in the MATLAB current folder as `html_example.html`.

Create a Word report.

```
import mlreportgen.dom.*;
rpt = Document('HTMLReport', 'docx');
```

Append the HTML file content to the document.

```
addHTMLFile(rpt, 'html_example.html');
```

Generate the Word report.

```
close(rpt);
rptview(rpt.OutputPath);
```

## Tips

- MATLAB Report Generator `mlreportgen.dom.HTML` or `mlreportgen.dom.HTMLFile` objects typically cannot accept the raw HTML output of third-party applications, such as Microsoft Word, that export native documents as HTML markup. In these cases, your Report API report generation program can use the `mlreportgen.utils.html2dom.prepHTMLString` and `mlreportgen.utils.html2dom.prepHTMLFile` functions to prepare the raw HTML for use with the `mlreportgen.dom.HTML` or `mlreportgen.dom.HTMLFile` objects. Typically, your program will have to further process the prepared HTML to remove valid but undesirable objects, such as line feeds that were in the raw content.
- By default, the DOM API uses a base font size of 12 points to convert `em` units to actual font sizes. For example, a font size specified as `2em` converts to 24 points. To specify a different base font size, add your content to a report by using an `mlreportgen.dom.HTML` object. Set the `EMBaseFontSize` property of the object to the base font size. For example, if you set the `EMBaseFontSize` property to 14, a font size of `2em` converts to 28 points.

## See Also

`mlreportgen.dom.HTML` | `mlreportgen.dom.HTMLFile` |  
`mlreportgen.utils.html2dom.prepHTMLFile` |  
`mlreportgen.utils.html2dom.prepHTMLString`

## Topics

“Convert HTML Content to a DOM Object” on page 13-105

“Prepare HTML for Conversion to a DOM Object” on page 13-108

“Requirements for Converting HTML to a DOM Object” on page 13-110

**Introduced in R2015a**

## append

**Class:** `mlreportgen.dom.Document`

**Package:** `mlreportgen.dom`

Append DOM or MATLAB object to document

### Syntax

```
domObjOut = append(docObj, textContent)
domObjOut = append(docObj, listContent)
domObjOut = append(docObj, tableContent)
domObjOut = append(docObj, paraObjpageLayoutObj)
domObjOut = append( ____, styleName)
```

```
domObjOut = append(docObj, domObj)
```

### Description

`domObjOut = append(docObj, textContent)` appends text or numbers to a document and returns a text object. In Microsoft Word and PDF output, the text is wrapped in a paragraph because Word and PDF do not permit unwrapped text to be added to the body of a document. In HTML output, the text is not wrapped in a paragraph.

`domObjOut = append(docObj, listContent)` appends an unordered list and returns an unordered list object.

`domObjOut = append(docObj, tableContent)` appends a table and returns a table object.

`domObjOut = append(docObj, paraObjpageLayoutObj)` appends a paragraph, starts a new page layout section whose properties are specified by the `pageLayoutObject`, and returns a paragraph object.

`domObjOut = append( ____, styleName)` appends the specified content, using the specified style.

`domObjOut = append(docObj, domObj)` appends a DOM object to the document and returns that object.

### Input Arguments

**docObj — Document to which to append content**

`mlreportgen.dom.Document` object

Document to which to append content, specified as an `mlreportgen.dom.Document` object.

**textContent — Text to append to document**

`string` | character vector

Text to append to document, specified as a string or character vector. The text object is wrapped in a paragraph object and the paragraph is appended to the document. The text is wrapped in a



paragraph to be consistent with Microsoft Word, which does not allow text to be added to the body of a document. For HTML, text wrapping can cause unexpected behavior.

### **listContent — List object to append to document**

ordered list object | unordered list object | array

List object to append to document, specified as an ordered list, unorderedlist, or an array. If the input is a 1D horizontal array of double values or strings, or a 1D categorical array, an unorderedlist object is created and that list object is appended it to the document.

### **tableContent — Table object to append to document**

array | header array and body array | MATLAB table

Table object to append to document, specified as one of:

- 2D array of double values — Appends and returns a Table object
- 2D array of strings — Appends and returns a Table object
- 2D categorical array — Appends and returns a Table object
- Cell array of strings for the table header and a numeric, cell, or categorical array for the table body — Appends and returns a FormalTable object
- MATLAB table — Appends and returns a MATLABTable object

### **paraObj — Paragraph to append to document**

paragraph object

Paragraph to append to document, specified as a paragraph object. It also starts a new page layout section with properties specified by the pageLayoutObj input.

### **pageLayoutObj — Page layout to apply to the appended page layout section**

paragraph object

Page layout to apply to the appended page layout section, specified as a PageLayout object.

### **styleName — Style to apply to input**

style

Style to apply to text, table or list input.

### **domObj — DOM object to append to document**

mlreportgen.dom object

DOM object to append to document, specified as any of these mlreportgen.dom objects:

- Container
- CustomElement
- DOCXPageLayout
- ExternalLink
- FormalTable
- Group
- HorizontalRule
- HTML

- HTMLFile
- Image
- InternalLink
- LineBreak
- LinkTarget
- MATLABTable
- NumPages
- OrderedList
- Page
- PageBreak
- PageRef
- Paragraph
- PDFPageLayout
- RawText
- StyleRef
- Table
- Text
- UnorderedList

## Output Arguments

### **domObjOut** — Appended object returned

DOM object

Appended object returned. The type of object depends on the second input type.

## Examples

### **Append an Ordered List Object**

Create an `OrderedList` object and append it to a report.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'html');

ol = OrderedList({'first step' 'second step' 'last step'});
append(d,ol);

close(d);
rptview('mydoc', 'html');
```

### **Specify a Style for Appended Text**

Use the `Word Title` style for the text.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'docx');
```

```
append(d, 'This Is a Title', 'Title');
close(d);
rptview('mydoc', 'docx');
```

### Append a MATLAB Table

```
% Create a MATLAB table named patients from workspace variables.
load patients;
BloodPressure = [Systolic Diastolic];
patients = table(Gender, Age, Smoker, BloodPressure);
patients.Properties.RowNames = LastName;

% Sort the table based on the Age variable.
sorted = sortrows(patients, 'Age');

% Create a report with the sorted patients table
rpt = mlreportgen.dom.Document('MyFileName', 'pdf');
append(rpt, sorted);
close(rpt);

% Show the PDF report in the viewer
rptview(rpt.OutputPath);
```

### Append a Cell Array as a Table

```
import mlreportgen.dom.*;
d = Document('mydoc');
table = append(d, {'row 1 - col 1' 'row 1 - col 2';...
    'row 2 - col 1' 'row 2 - col 2'});
table.Style = {Border('double'), ColSep('solid'), RowSep('solid')};
close(d);
rptview('mydoc', 'html');
```

## See Also

[mlreportgen.dom.Document](#) | [mlreportgen.dom.MATLABTable](#)

### Topics

“Add Content to a Report” on page 13-10

### Introduced in R2014b

## close

**Class:** `mlreportgen.dom.Document`

**Package:** `mlreportgen.dom`

Close document

### Syntax

```
close(docObj)
```

### Description

`close(docObj)` closes a document. Once a document is closed, you can no longer append content to it. Closing the document outputs any remaining content, such as remaining template text.

### Examples

#### Close a Document

Close the `myReport` document.

```
import mlreportgen.dom.*;
myReport = Document('mydoc', 'html');

append(myReport, Paragraph('This is an introduction'));

close(myReport);
rptview('mydoc', 'html');
```

### Input Arguments

#### **docObj** — Document to close

`mlreportgen.dom.Document` object

Document to close, specified as an `mlreportgen.dom.Document` object.

### See Also

`open` | `mlreportgen.dom.Document`

### Topics

“Add Content to a Report” on page 13-10

### Introduced in R2014b

# createAutoNumberStream

**Class:** mlreportgen.dom.Document

**Package:** mlreportgen.dom

Create numbering stream

## Syntax

```
streamOut = createAutoNumberStream(docObj, streamName)
streamOut = createAutoNumberStream(docObj, streamName, streamType)
streamOut = createAutoNumberStream(docObj, streamName, streamType,
initialValue)
```

## Description

`streamOut = createAutoNumberStream(docObj, streamName)` creates a numbering stream using Arabic numbers and an initial value of 0.

`streamOut = createAutoNumberStream(docObj, streamName, streamType)` creates a numbering stream using the specified type of characters (Arabic numbers, alphabetic, or Roman numerals) and an initial value corresponding to 0 (for example, a or i).

`streamOut = createAutoNumberStream(docObj, streamName, streamType, initialValue)` creates a numbering stream using the specified type of characters (Arabic numbers, alphabetic, or Roman numerals) and specified initial value.

## Examples

### Create a Numbering Stream for Chapter Heading

```
import mlreportgen.dom.*;
myReport = Document('mydoc', 'html');

chapStream = createAutoNumberStream(myReport, 'chapter', 'I');
for i=1:5
    p = Paragraph('Chapter ');
    p.Style = {CounterInc('chapter')};
    p.WhiteSpace = 'pre';
    append(p, AutoNumber('chapter'));
    append(myReport, p);
end

close(myReport);
rptview(myReport.OutputPath);
```

## Input Arguments

**docObj** — Document to apply numbering stream to

mlreportgen.dom.Document object

Document to apply numbering stream to, specified as an `mlreportgen.dom.Document` object.

**streamName — Name of numbering stream**

character vector

Consider using a name that indicates the kinds of document element (for example, a chapter heading) that you expect to apply the stream to.

**streamType — Type of numbering stream characters**

'n' (default) | 'N' | 'a' | 'A' | 'i' | 'I'

Use one of these letters to specify the type of characters to use for the numbering values.

- 'n' — Arabic numerals (you can also use 'N')
- 'a' — Lowercase alphabetic letters (a, b, c,...)
- 'A' — Uppercase alphabetic letters (A, B, C,...)
- 'i' — Lowercase Roman numerals (i, ii, iii,...)
- 'I' — Uppercase Roman numerals (I, II, III,...)

**initialValue — Starting value for a numbering stream**

number

Use a number, regardless of the type of stream. The initial value used by the stream depends on the type of stream. For example, if you set `initialValue` to 0:

- Arabic numeral stream — 0
- Alphabetic stream — a or A
- Roman numerals stream — i or I

Data Types: `double`

## Output Arguments

**streamOut — Numbering stream**

`mlreportgen.dom.AutoNumberStream` object

A numbering stream, represented by an `mlreportgen.dom.AutoNumberStream` object.

## Tips

When you append an `mlreportgen.dom.AutoNumber` object, specify a numbering stream.

## See Also

`mlreportgen.dom.Document | getAutoNumberStream`

## Topics

“Automatically Number Document Content” on page 13-101

## Introduced in R2014b

# mlreportgen.dom.Document.createTemplate

**Class:** mlreportgen.dom.Document

**Package:** mlreportgen.dom

Create DOM template file

## Syntax

```
templatePath = mlreportgen.dom.Document.createTemplate(path)
templatePath = mlreportgen.dom.Document.createTemplate(path,type)
```

## Description

`templatePath = mlreportgen.dom.Document.createTemplate(path)` creates a copy of the default DOM default template at the specified location and returns the full path of the new template. The path argument must include a file name extension that indicates the type of template to create.

`templatePath = mlreportgen.dom.Document.createTemplate(path,type)` creates a copy of the DOM default template of the specified type at the specified location and returns the full path of the new template.

## Examples

### Create a Word Template

Create a Microsoft Word template file in the current folder.

```
mlreportgen.dom.Document.createTemplate('MyWordTemplate','docx');
```

## Input Arguments

### path — Path of new template file

character vector | string scalar

Path, including the file name, of the new template file, specified as a character vector or string scalar. If you use the path argument without the type argument, include the file name extension:

- .htmtx or .htm for HTML
- .dotx or .docx for Word
- .pdftx or .pdf for PDF
- .htmt or .html for single-file HTML

If you use both the path and type arguments the mlreportgen.dom.Document.createTemplate method ignores the file name extension in the path argument.

### type — Type of template

'html' | 'docx' | 'pdf' | 'html-file'

Type of template, specified as:

- 'html' for HTML.
- 'docx' for Word.
- 'pdf' for PDF.
- 'html-file' for single-file HTML. HTML output consists of a single file that contains the text, style sheets, JavaScript®, and images for the report.

## Output Arguments

### **templatePath** — Full path of template copy

character vector

Full path of the template copy, specified as a character vector. The path includes a file name extension:

- .htm1x for HTML
- .dotx for Word
- .pdf1x for PDF
- .html for single-file HTML

## See Also

[zipTemplate](#) | [unzipTemplate](#) | [mlreportgen.dom.Template](#) | [mlreportgen.dom.Document](#)

## Topics

“Create a Microsoft Word Template” on page 13-119

“Create an HTML or PDF Template” on page 13-130

## Introduced in R2014b



# fill

**Class:** mlreportgen.dom.Document

**Package:** mlreportgen.dom

Fill document holes with generated content

## Syntax

```
fill(form)
```

## Description

`fill(form)` fills the holes in a DOM-based form with generated content. Use this method with a class you derive from any of these classes:

- mlreportgen.dom.Document
- mlreportgen.dom.DocumentPart
- mlreportgen.dom.DOCXPageHeader
- mlreportgen.dom.DOCXPageFooter
- mlreportgen.dom.PDFPageHeader
- mlreportgen.dom.PDFPageFooter
- mlreportgen.dom.Template

---

**Note** Use this method only with derived classes. Invoking this method on an instance of a DOM class causes an error.

---

This method assumes that the derived class, for each hole in an instance's template, defines a method having this signature:

```
fillHoleId(d)
```

`HoleId` is the ID of a hole defined in the document's template. `d` is an instance of the derived class. The fill method moves from the first hole in the document to the last, invoking the corresponding `fillHoleId` method at each hole. This way, you can define methods that fill the holes without looping. The fill method moves from hole to hole to fill the template.

## Input Arguments

**form — Form to fill**

character vector

Form whose holes to fill, specified as a character vector.

## Examples

### Add a fill Method to Invoke Hole-Specific fill Methods

This example shows how to define a report that fills a `CustomerName` hole in a Word template.

Create a template that has a `CustomerName` hole. This example assumes that there is a Word template called `CustomerLetter.docx`.

In a file, create a report class derived from `mlreportgen.dom.Document`. From the MATLAB toolstrip, select **New > Class** and define the class. For example:

```
classdef MyReport < mlreportgen.dom.Document
    %MYREPORT defines a customize letter to customers
    %
    % rpt = MyReport('mydoc','docx','CustomerLetter');
    % rpt.CustomerName = 'Smith';
    % fill(rpt);

    properties
        CustomerName;
    end

    methods
        function rpt = MyReport(filename,type,template)
            rpt = rpt@mlreportgen.dom.Document(filename,type,template);
        end

        function fillCustomerName(rpt)
            append(rpt,rpt.CustomerName);
        end
    end
end
```

Use the report.

```
rpt = MyReport('mydoc','docx','CustomerLetter');
rpt.CustomerName = 'Mr. Smith';
fill(rpt);
```

### Tips

In the derived class, define `fill` methods to insert content for each hole in the template. Use this signature:

```
fillHOLE_ID(docObj);
```

`HOLE_ID` is the ID of a hole defined by the template that the document uses, and `docObj` is an instance of the derived class. When invoked on a derived `Document` object, the `fill` method moves from the first hole in the document to the last, invoking the corresponding `fillHOLE_ID` method at each hole. This approach eliminates the need for additional code to loop through the holes in a template.

### See Also

`mlreportgen.dom.Document` | `moveToNextHole`

**Topics**

"Add Content to a Report" on page 13-10

**Introduced in R2014b**

## getAutoNumberStream

**Class:** `mlreportgen.dom.Document`

**Package:** `mlreportgen.dom`

Return numbering stream

### Syntax

```
autoNumStreamOut = getAutoNumberStream(docObj, streamName)
```

### Description

`autoNumStreamOut = getAutoNumberStream(docObj, streamName)` returns the specified numbering stream used by the document.

### Examples

#### Return a Numbering Stream

```
import mlreportgen.dom.*;
myReport = Document('mydoc', 'html');

createAutoNumberStream(myReport, 'chapterNum', 'I', 1);
streamOut = getAutoNumberStream(myReport, 'chapterNum')
```

`streamOut =`

AutoNumberStream with properties:

```
StreamName: 'chapterNum'
CharacterType: 'roman'
CharacterCase: 'upper'
InitialValue: 1
Tag: 'dom.AutoNumberStream:500'
Id: '500'
```

### Input Arguments

#### **docObj** — Document that uses numbering stream

`mlreportgen.dom.Document` object

Document that uses the numbering stream, specified as an `mlreportgen.dom.Document` object.

#### **streamName** — Name of numbering stream to return

character vector

Name of the numbering stream to return, specified as a character vector.

## Output Arguments

### **autoNumStreamOut — Numbering stream used by document**

`mlreportgen.dom.AutoNumberStream` object

Numbering stream used by the document, represented by an `mlreportgen.dom.AutoNumberStream` object.

## See Also

`createAutoNumberStream` | `mlreportgen.dom.AutoNumberStream` | `mlreportgen.dom.AutoNumber`

## Topics

“Automatically Number Document Content” on page 13-101

## Introduced in R2014b

## mlreportgen.dom.Document.getCoreProperties

**Class:** mlreportgen.dom.Document

**Package:** mlreportgen.dom

Get document or template core properties

### Syntax

```
corePropertiesOut = mlreportgen.dom.Document.getCoreProperties(path)
```

### Description

`corePropertiesOut = mlreportgen.dom.Document.getCoreProperties(path)` specifies the core OPC properties for the document or template having the specified path.

### Examples

#### Return Core Properties of a Document

```
import mlreportgen.dom.*;
myReport = Document('mydoc', 'docx');

append(myReport, 'Hello world');

close(myReport);
coreProps = Document.getCoreProperties('mydoc.docx')

coreProps =
```

CoreProperties with properties:

```
    Category: []
ContentStatus: []
    Creator: 'MathWorks'
  Description: []
  Identifier: []
    Keywords: []
    Language: []
LastModifiedBy: ''
    Revision: '2'
    Subject: []
    Title: ''
    Version: []
    Tag: 'dom.CoreProperties:203'
    Id: '203'
```

### Input Arguments

**path** — Path to document or template

character vector

Path to the document or template, specified as a character vector.

## Output Arguments

### **corePropertiesOut** — Core properties of document or template

`mlreportgen.dom.CoreProperties` object

Core properties of the document or template, represented by an `mlreportgen.dom.CoreProperties` object.

## See Also

`mlreportgen.dom.Document.setCoreProperties` |

`mlreportgen.dom.Document.getOPCMainPart` | `mlreportgen.dom.OPCPart`

## Topics

“Output Types and Report Generator Packages” on page 13-14

## Introduced in R2014b

## mlreportgen.dom.Document.getImageDirectory

**Class:** mlreportgen.dom.Document

**Package:** mlreportgen.dom

Get image folder of document

### Syntax

```
imageDirectory = mlreportgen.dom.Document.getImageDirectory(path,type)
```

### Description

`imageDirectory = mlreportgen.dom.Document.getImageDirectory(path,type)` gets the image folder of a document or template package located at the specified path and of the specified type (Microsoft Word or HTML). This is a static method. Invoke it on the Document class.

### Examples

#### Get the Image Folder

Suppose that the main image folder of an HTML document named `MyDoc.htm` resides in `images` under the root of the package. To get the path, enter:

```
mlreportgen.dom.Document.getImageDirectory('MyDoc','html');
```

```
ans =
```

```
/images
```

### Input Arguments

#### **path** — Path of document or template package

character vector

Path of the document or template package

#### **type** — Type of document or template

'docx' | 'html'

Type document or template. For a Word document or template, specify 'docx'. For an HTML document or template, specify 'html'.

### Output Arguments

#### **imageDirectory** — Image folder of document

character vector

The path of the image folder for the package.



## **See Also**

`mlreportgen.dom.Document.getImagePrefix` |  
`mlreportgen.dom.Document.setCoreProperties` |  
`mlreportgen.dom.Document.getOPCMainPart` | `mlreportgen.dom.OPCPart`

## **Topics**

“Output Types and Report Generator Packages” on page 13-14

## **Introduced in R2014b**

## mlreportgen.dom.Document.getImagePrefix

**Class:** mlreportgen.dom.Document

**Package:** mlreportgen.dom

Get generated image name prefix

### Syntax

```
imagePrefix = mlreportgen.dom.Document.getImagePrefix(path,type)
```

### Description

`imagePrefix = mlreportgen.dom.Document.getImagePrefix(path,type)` gets the image name prefix of a document or template package located at the specified path and of the specified type (Microsoft Word or HTML). The DOM interface uses the prefix when generating internal names of images appended to the document. This is a static method. Invoke it on the Document class.

### Examples

#### Get the Image Name Prefix

Suppose that the image name prefix of an HTML document named `MyDoc.htmx` is `image`. To get the image name prefix, enter:

```
mlreportgen.dom.Document.getImagePrefix('MyDoc','html');
```

```
ans =
```

```
image
```

### Input Arguments

**path** — Path of document or template package

character vector

Path of the document or template package

**type** — Type of document or package

'docx' | 'html'

Type document or template. For a Word document or template, specify 'docx'. For an HTML document or template, specify 'html'.

### Output Arguments

**imagePrefix** — Image name prefix

character vector

The generated image name prefix.

## **See Also**

`mlreportgen.dom.Document.getImageDirectory` |  
`mlreportgen.dom.Document.setCoreProperties` |  
`mlreportgen.dom.Document.getOPCMainPart` | `mlreportgen.dom.OPCPart`

## **Topics**

“Output Types and Report Generator Packages” on page 13-14

## **Introduced in R2014b**

## getMainPartPath

**Class:** mlreportgen.dom.Document

**Package:** mlreportgen.dom

Return path of main part of document output package

### Syntax

```
pathOut = getMainPartPath(docObj)
```

### Description

`pathOut = getMainPartPath(docObj)` returns the full path of the main part of the output package of the specified document. The main part is the file that contains the XML or HTML markup for a document.

### Examples

#### Get Path to Main Part of Output Package

This code returns the path to the main part of an HTML document named `MyReport`. The main part is named `index.html` and it is in the root of the `MyReport` package. This example assumes that there is a `reports` folder on the `S:` drive.

```
d = mlreportgen.dom.Document('S:\reports\MyReport', 'html');  
d.PackageType='unzipped';  
getMainPartPath(d)
```

```
's:\reports\MyReport\index.html'
```

### Input Arguments

#### **docObj** — Document that contains main part

mlreportgen.dom.Document object

Document that contains the main part, specified as an mlreportgen.dom.Document object.

### Output Arguments

#### **pathOut** — Path of main part of document output package

character vector

Path of the main part of document output package.

### See Also

mlreportgen.dom.Document.setCoreProperties |  
mlreportgen.dom.Document.getOPCMainPart | mlreportgen.dom.OPCPart

**Topics**

“Output Types and Report Generator Packages” on page 13-14

**Introduced in R2014b**

## mlreportgen.dom.Document.getOPCMainPart

**Class:** mlreportgen.dom.Document

**Package:** mlreportgen.dom

Return main part of document, document part, or template

### Syntax

```
partOut = mlreportgen.dom.Document.getOPCMainPart(path)
partOut = mlreportgen.dom.Document.getOPCMainPart(path, docType)
```

### Description

`partOut = mlreportgen.dom.Document.getOPCMainPart(path)` returns the path of the main part (file) of a package for a document, document part, or template, based on the specified path. The returned path is relative to the root directory of the package, which is symbolized by a forward slash (/). The main part is the file that contains the document or template XML or HTML markup.

`partOut = mlreportgen.dom.Document.getOPCMainPart(path, docType)` returns the relative path of the main part of the output package of the specified type (Microsoft Word or HTML) of document, document part, or template.

### Examples

#### Get Path to Main Part of a Document Package

The example returns the path to the main part of an HTML document named `myDoc.htmx`. The main part is named `root.html`, which is in the top-level folder of the package.

```
import mlreportgen.dom.*;
myDocument = Document('myDoc', 'html');

append(myDocument, 'Hello world');

close(myDocument);
mlreportgen.dom.Document.getOPCMainPart('MyDoc.htmx', 'html')

ans =

/root.html
```

### Input Arguments

#### **path** — Path of document

character vector

If you use the `path` argument without the `docType` argument, include the `.docx` or `.htm` extension.

If you use both the `path` and `docType` arguments, `getOPCMainPart` appends an extension of the appropriate type (`.docx` or `.htm`).

**docType — Type of document, document part, or template**

`'docx' | 'html'`

Type of document, document part, or template, specified as a character vector.

## Output Arguments

**partOut — Path of main part of a package**

character vector

The path to the main part of the document, document part, or template. The returned path is relative to the root directory of the package, which is symbolized by a forward slash (/).

## Tips

The `getOPCMainPart` method is a static method. Invoke it on the `Document` class, rather than on an instance of the `Document` class or on a class derived from the `Document` class.

## See Also

`mlreportgen.dom.Document.setCoreProperties` | `getMainPartPath` | `mlreportgen.dom.OPCPart`

## Topics

“Output Types and Report Generator Packages” on page 13-14

## Introduced in R2014b

## moveToNextHole

**Class:** `mlreportgen.dom.Document`

**Package:** `mlreportgen.dom`

Move document append point to next template hole

### Syntax

```
holeID = moveToNextHole(docObj)
```

### Description

`holeID = moveToNextHole(docObj)` copies to the output document any text between the current hole and the next hole in the document template. DOM creates an `mlreportgen.dom.RawText` object for the text. This method makes the next hole the current hole and returns the ID of that hole.

### Examples

#### Move to Next Hole

```
import mlreportgen.dom.*;
myReport = Document('myDoc', 'docx');

moveToNextHole(myReport);
```

### Input Arguments

#### **docObj** — Document

`mlreportgen.dom.Document` object

Document in which to move the append point to the next hole.

### Output Arguments

#### **holeID** — Template hole ID

hole ID

The ID of the template hole that the method moves to (the new current hole).

Data Types: `char`

### Tips

The first time you invoke the `moveToNextHole` method, the DOM copies to the output document all of the text up to the first hole in the template. Use `Document.append` methods to add content to the output document to fill the first hole. The next time you invoke `moveToNextHole`, the DOM copies to the output document all the text between the first and second hole in the template. Then use



Document .append methods to fill in the second hole. In this way, you can successively fill all of the holes in a document.

## **See Also**

`mlreportgen.dom.Document | fill`

## **Topics**

“Add Content to a Report” on page 13-10

**Introduced in R2014b**

## open

**Class:** `mlreportgen.dom.Document`

**Package:** `mlreportgen.dom`

Open document

### Syntax

```
open(docObj)
```

### Description

`open(docObj)` opens a document for appending content.

### Examples

#### Open a Document

```
import mlreportgen.dom.*;
myReport = Document('myDoc', 'html');

open(myReport);
```

### Input Arguments

#### **docObj** — Document to open

`mlreportgen.dom.Document` object

Document to open, specified as an `mlreportgen.dom.Document` object.

### Tips

- After you open a document, you can no longer change its generated document type or the template.
- The `append` method for a document opens a document if the document is not already opened. Therefore, you rarely need to use the `open` method.

### See Also

`close` | `mlreportgen.dom.Document`

#### Topics

“Add Content to a Report” on page 13-10

**Introduced in R2014b**

## package

**Class:** mlreportgen.dom.Document

**Package:** mlreportgen.dom

Add OPC part files to document package

### Syntax

```
partOut = package(docObj, opcPart)
```

### Description

`partOut = package(docObj, opcPart)` adds a file specified by an OPC part object to the OPC package of a document.

### Examples

#### Add Files to a Document Package

This example shows how to use the `package` method to add special browser processing code. In this example, the `processData.js` file operates on the `data.json` file. This example assumes that there are `data.json` and `processData.js` files in the current folder.

```
import mlreportgen.dom.*;
myReport = Document('myDoc', 'html');

package(myReport, OPCPart('/data/data.json', 'data.json'));
package(myReport, OPCPart('/js/processData.js', 'processData.js'))

close(myReport);
```

### Input Arguments

#### **docObj** — Document OPC package to add files to

mlreportgen.dom.Document object

Document OPC package to add files to, specified as an mlreportgen.dom.Document object.

#### **opcPart** — OPC part that specifies file to add to OPC package

mlreportgen.dom.OPCPart object

Define an OPCPart object to specify the files to add.

### Output Arguments

#### **partOut** — Added OPC part file

mlreportgen.dom.OPCPart object

Added OPC part file, represented by an mlreportgen.dom.OPCPart object.

### **See Also**

`zipTemplate` | `unzipTemplate` | `mlreportgen.dom.Document.getCoreProperties` | `mlreportgen.dom.Document.getOPCMainPart` | `mlreportgen.dom.OPCPart`

### **Topics**

“Output Types and Report Generator Packages” on page 13-14

### **Introduced in R2014b**

# mlreportgen.dom.Document.setCoreProperties

**Class:** mlreportgen.dom.Document

**Package:** mlreportgen.dom

Set OPC core properties of output document or template

## Syntax

```
corePropertiesOut = mlreportgen.dom.Document.setCoreProperties(path,  
corePropertiesObj)
```

## Description

`corePropertiesOut = mlreportgen.dom.Document.setCoreProperties(path, corePropertiesObj)` sets the core OPC property values of the document or template having the specified path.

## Examples

### Set OPC Core Properties for a Document Package

This example shows how to use `setCoreProperties` to apply core property settings to a report.

```
import mlreportgen.dom.*;  
myReport = Document('mydoc', 'docx');  
  
append(myReport, 'Hello world');  
close(myReport);  
coreProps = Document.getCoreProperties('mydoc.docx');  
coreProps.Title = 'MATLAB Example';  
Document.setCoreProperties('mydoc.docx', coreProps)
```

In Windows Explorer, if you navigate to the `mydoc.docx` file, you can see that the `Title` field says `MATLAB Example`.

## Input Arguments

### **path** — Path of document, document part, or template

character vector

Path of document, document part, or template, specified as a character vector.

### **corePropertiesObj** — OPC core properties to use

mlreportgen.dom.CoreProperties object

OPC core properties to use, specified as an `mlreportgen.dom.CoreProperties` object.

## Output Arguments

### **corePropertiesOut — OPC core properties**

`mlreportgen.dom.CoreProperties` object

OPC core properties, represented by an `mlreportgen.dom.CoreProperties` object.

## See Also

`mlreportgen.dom.Document.getCoreProperties` |

`mlreportgen.dom.Document.getOPCMainPart` | `mlreportgen.dom.CoreProperties`

## Topics

“Output Types and Report Generator Packages” on page 13-14

**Introduced in R2014b**

# append

**Class:** `mlreportgen.dom.ExternalLink`

**Package:** `mlreportgen.dom`

Append custom element to external link

## Syntax

```
textObjOut = append(externalLinkObj, text)
textObjOut = append(externalLinkObj, text, styleName)
domObjOut = append(externalLinkObj, domObj)
```

## Description

`textObjOut = append(externalLinkObj, text)` appends a `Text` object constructed from the specified text to the link.

`textObjOut = append(externalLinkObj, text, styleName)` appends text using the specified style.

`domObjOut = append(externalLinkObj, domObj)` appends a `Text`, `Image`, or `CustomElement` object to the link.

## Input Arguments

**externalLinkObj** — External link object to append custom element to

`mlreportgen.dom.ExternalLink` object

External link object to append custom element to, specified as an `mlreportgen.dom.ExternalLink` object.

**text** — Text to append

character vector

Text to append, specified as a character vector.

**styleName** — Name of style to apply to appended text

character vector

The style to use with the appended text. The style defines the appearance of the document element in the output document.

Use a style that is defined in the style sheet of the template of the document you append content to.

**domObj** — DOM object to append

`mlreportgen.dom.Text` object | `mlreportgen.dom.Image` object |  
`mlreportgen.dom.CustomElement` object

DOM object to append, specified as an `mlreportgen.dom.Text`, `mlreportgen.dom.Image`, or `mlreportgen.dom.CustomElement` object.

## Output Arguments

### **textObjOut — Text appended to external link**

`mlreportgen.dom.Text` object

Text appended to an external link, represented by an `mlreportgen.dom.Text` object.

### **domObjOut — DOM object appended to external link**

`mlreportgen.dom.Text` object | `mlreportgen.dom.Image` object |  
`mlreportgen.dom.CustomElement` object

External link with appended content, returned as a DOM object of the same class as the input argument DOM object.

## See Also

`mlreportgen.dom.ExternalLink` | `mlreportgen.dom.LinkTarget` |  
`mlreportgen.dom.InternalLink`

## Topics

“Create Links” on page 13-82

## Introduced in R2014b



# appendFooterRow

**Class:** mlreportgen.dom.FormalTable

**Package:** mlreportgen.dom

Append row to table footer

## Syntax

```
rowObjOut = appendFooterRow(tableObj, rowObj)
```

## Description

rowObjOut = appendFooterRow(tableObj, rowObj) appends a row of table entries to the footer of a table.

## Examples

### Append a Table Footer

Create, format, and append a formal table.

```
import mlreportgen.dom.*;
myReport = Document('myDoc', 'html');

table = FormalTable({'row1 - col1' 'row1 - col2 ';...
    'row2 - col1' 'row2 - col2'});
table.Style = {Border('double'), ColSep('solid'), RowSep('solid')};
append(myReport, table);
```

Create a row (and its entries) for the footer. Use bold text for the text in the row.

```
rowForFooter = TableRow();
rowForFooter.Style = {Bold(true)};
col1Title = TableEntry('Column 1 footer');
col2Title = TableEntry('Column 2 footer');
append(rowForFooter, col1Title);
append(rowForFooter, col2Title);
```

Append the footer row and display the report.

```
footerRow = appendFooterRow(table, rowForFooter);

close(myReport);
rptview('myDoc', 'html');
```

## Input Arguments

### tableObj — Table

mlreportgen.dom.FormalTable object

Table that contains the footer to append a row to.

**rowObj** — Row to append to table footer

`mlreportgen.dom.TableRow` object

Row to append to the table footer, specified as an `mlreportgen.dom.TableRow` object.

**Output Arguments****rowObjOut** — Row appended to table footer

`mlreportgen.dom.TableRow` object

Row appended to the table footer, represented by an `mlreportgen.dom.TableRow` object.

**See Also**

`mlreportgen.dom.Table` | `mlreportgen.dom.FormalTable`

**Topics**

“Create Informal Tables” on page 13-64

**Introduced in R2014b**

## appendHeaderRow

**Class:** mlreportgen.dom.FormalTable

**Package:** mlreportgen.dom

Append row to table header

### Syntax

```
rowObjOut = appendHeaderRow(tableObj, rowObj)
```

### Description

rowObjOut = appendHeaderRow(tableObj, rowObj) appends a row of table entries to the header of this table.

### Examples

#### Append a Table Header

Create a formal table.

```
import mlreportgen.dom.*;
myReport = Document('myDoc', 'html');

table = FormalTable({'row1 - col1' 'row1 - col2 ';...
    'row2 - col1' 'row2 - col2'});
table.Style = {Border('double'), ColSep('solid'), RowSep('solid')};
append(myReport, table);
```

Create a row for the header.

```
rowForHeader = TableRow();
col1Title = TableEntry('Column 1 header');
col2Title = TableEntry('Column 2 header');
append(rowForHeader, col1Title);
append(rowForHeader, col2Title);
```

Append the header row and display the report.

```
headerRow = appendHeaderRow(table, rowForHeader);

close(myReport);
rptview('myDoc', 'html');
```

### Input Arguments

#### tableObj — Table

mlreportgen.dom.FormalTable object

Table that contains the header to append a row to.

**rowObj** — Row to append to table header`mlreportgen.dom.TableRow` object

Row to append to the table header, specified as an `mlreportgen.dom.TableRow` object.

**Output Arguments****rowObjOut** — Row appended to table header`mlreportgen.dom.TableRow` object

Row appended to table header, represented by an `mlreportgen.dom.TableRow` object.

**See Also**`mlreportgen.dom.Table` | `mlreportgen.dom.FormalTable`**Topics**

“Create Informal Tables” on page 13-64

**Introduced in R2014b**

# append

**Class:** `mlreportgen.dom.Group`

**Package:** `mlreportgen.dom`

Add DOM object to group

## Syntax

```
domObjOut = append(groupObj, domObj)
```

## Description

`domObjOut = append(groupObj, domObj)` appends a DOM object to a group. Use groups to append a set of document elements together.

## Examples

### Append a Paragraph to a Group

```
import mlreportgen.dom.*;
myReport = Document('myDoc', 'html');

x = 0:pi/100:2*pi;
y = sin(x);
plot1 = plot(x,y);
saveas(plot1, 'plot1.png')

plotimage = Image('plot1.png');
para = Paragraph('Value of the sine function from 0 to 2pi');
groupForPlot = Group();
append(groupForPlot, para);
append(groupForPlot, plotimage);
append(myReport, groupForPlot);

close(myReport);
rptview('myDoc', 'html');
```

## Input Arguments

### **groupObj** — Group object to append DOM object to

`mlreportgen.dom.Group` object

Group object to append the DOM object to, specified as an `mlreportgen.dom.Group` object.

### **domObj** — DOM document element object to append

DOM object

You can append the following DOM objects:

- `mlreportgen.dom.CustomElement`
- `mlreportgen.dom.DocumentPart`
- `mlreportgen.dom.FormalTable`
- `mlreportgen.dom.Group`
- `mlreportgen.dom.ExternalLink`
- `mlreportgen.dom.HTML`
- `mlreportgen.dom.HTMLFile`
- `mlreportgen.dom.Image`
- `mlreportgen.dom.InternalLink`
- `mlreportgen.dom.LinkTarget`
- `mlreportgen.dom.OrderedList`
- `mlreportgen.dom.Paragraph`
- `mlreportgen.dom.RawText`
- `mlreportgen.dom.Table`
- `mlreportgen.dom.TemplateHole`
- `mlreportgen.dom.Text`
- `mlreportgen.dom.UnorderedList`

## Output Arguments

### **domObjOut** — Appended document object

DOM object

Appended document object, returned as the corresponding DOM object.

## See Also

`mlreportgen.dom.Group` | `mlreportgen.dom.Document` | `mlreportgen.dom.DocumentPart` | `mlreportgen.dom.DOCXPageLayout` | `mlreportgen.dom.PDFPageLayout`

## Topics

“Add Content to a Report” on page 13-10

## Introduced in R2014b

# append

**Class:** `mlreportgen.dom.HTML`

**Package:** `mlreportgen.dom`

Append HTML string to HTML object

## Syntax

```
htmlObjOut = append(htmlObj,htmlText)
htmlObjOut = append(htmlObj,htmlObjToAppend)
```

## Description

`htmlObjOut = append(htmlObj,htmlText)` converts HTML string into an HTML object and appends the object to `htmlObj`.

`htmlObjOut = append(htmlObj,htmlObjToAppend)` appends the `htmlObjToAppend` object to `htmlObj`.

## Input Arguments

**htmlObj** — HTML object to append content to

`mlreportgen.dom.HTML` object

HTML object to append content to, specified as an `mlreportgen.dom.HTML` object.

**htmlText** — HTML text

character vector

HTML text, specified as a character vector

Example: '`<p><b>Hello</b> <i style="color:green">World</i></p>`'

**htmlObjToAppend** — HTML object to append

`mlreportgen.dom.HTML` object

HTML object to append, specified as an `mlreportgen.dom.HTML` object.

## Output Arguments

**htmlObjOut** — HTML content appended to HTML object

`mlreportgen.dom.HTML` object

HTML content appended to an HTML object, returned as an `mlreportgen.dom.HTML` object.

## Examples

**Append HTML text to an HTML Object**

Create an HTML object from HTML text, to use for a Microsoft Word report.

```
import mlreportgen.dom.*;
rpt = Document('HTML2WordReport','docx');
htmlObj = HTML('<p><b>Hello</b> <i style="color:green">World</i></p>');
```

Append content to the HTML object. Append the HTML object to the document.

```
append(htmlObj,'<p>This is <u>me</u> speaking</p>');
append(rpt,htmlObj);
```

Generate the report.

```
close(rpt);
rptview(rpt.OutputPath);
```

**See Also**

[mlreportgen.dom.HTML](#) | [mlreportgen.dom.HTMLFile](#) | [addHTML](#)

**Topics**

“Convert HTML Content to a DOM Object” on page 13-105

**Introduced in R2015a**



# clone

**Class:** `mlreportgen.dom.HTML`

**Package:** `mlreportgen.dom`

Copy HTML object

## Syntax

```
clonedHTMLObj = clone(sourceHTMLObj)
```

## Description

`clonedHTMLObj = clone(sourceHTMLObj)` copies (clones) the specified HTML object, including its children.

## Input Arguments

**sourceHTMLObj** — HTML object to copy

`mlreportgen.dom.HTML` object

HTML object to copy, specified as an `mlreportgen.dom.HTML` object.

## Output Arguments

**clonedHTMLObj** — HTML object with appended content

an `mlreportgen.dom.HTML` object

HTML object with appended content, returned as an `mlreportgen.dom.HTML` object.

## Examples

### Copy an HTML Object

Create an HTML object from HTML text, to use for a Microsoft Word report.

```
import mlreportgen.dom.*;
rpt = Document('ClonedHTMLReport','docx');
htmlObj1 = HTML('<p><b>Hello</b> <i style="color:green">World</i></p>');
```

Append the HTML object to the report.

```
append(rpt,htmlObj1);
```

Copy the HTML object and append the copy to the report.

```
htmlObj2 = clone(htmlObj1);
append(rpt,htmlObj2);
```

Generate the report.

```
close(rpt);  
rptview(rpt.OutputPath);
```

### **See Also**

[mlreportgen.dom.HTML](#) | [mlreportgen.dom.HTMLFile](#)

### **Topics**

“Convert HTML Content to a DOM Object” on page 13-105

**Introduced in R2015a**

# append

**Class:** mlreportgen.dom.HTMLFile

**Package:** mlreportgen.dom

Append HTML to HTMLFile object

## Syntax

```
htmlObjOut = append(htmlFileObj,htmlText)
htmlObjOut = append(htmlFileObj,htmlObjToAppend)
```

## Description

`htmlObjOut = append(htmlFileObj,htmlText)` converts HTML string into an HTML object and appends the object to `htmlFileObj`.

`htmlObjOut = append(htmlFileObj,htmlObjToAppend)` appends the `htmlObjToAppend` object to `htmlFileObj`.

## Input Arguments

**htmlFileObj** — HTMLFile object to append content to

mlreportgen.dom.HTMLFile object

HTMLFile object to append content to, specified as an mlreportgen.dom.HTMLFile object.

**htmlText** — HTML text

character vector

HTML text, specified as a character vector.

Example: '`<p><b>Hello</b> <i style="color:green">World</i></p>`'

**htmlObjToAppend** — HTML object to append

mlreportgen.dom.HTML object

HTML object to append, specified as an mlreportgen.dom.HTML object.

**htmlFileObjToAppend** — HTMLFile object to append

mlreportgen.dom.HTMLFile object

HTMLFile object to append, specified as an mlreportgen.dom.HTMLFile object.

## Output Arguments

**htmlObjOut** — HTML content appended to HTMLFile object

mlreportgen.dom.HTML object

HTML content appended to an HTMLFile object, returned as an mlreportgen.dom.HTML object.

## Examples

### Append HTML Text to an HTMLFile Object

Create a text file named `myHTMLFile.html` and save it in the current folder. Add this text into the file:

```
<html>
<head>
<style>p {font-size:14pt;}</style>
</head>
<body>
<p style='white-space:pre'><b>Hello</b><i style='color:green'> World</i></p>
</body>
</html>
```

Create an `mlreportgen.dom.HTMLFile` object from an HTML file to use for a Microsoft Word report.

```
import mlreportgen.dom.*;
rpt = Document('MyReport','docx');
htmlFile = HTMLFile('myHTMLFile.html');
```

Append content to the `mlreportgen.dom.HTMLFile` object and append the object to the document.

```
append(htmlFile,'<p>This is <u>me</u> speaking</p>');
append(rpt,htmlFile);
```

Generate the report.

```
close(rpt);
rptview(rpt.OutputPath);
```

### See Also

`mlreportgen.dom.HTMLFile` | `mlreportgen.dom.HTML` |  
`mlreportgen.utils.html2dom.prepHTMLFile` |  
`mlreportgen.utils.html2dom.prepHTMLString`

### Topics

“Convert HTML Content to a DOM Object” on page 13-105

### Introduced in R2015a

## append

**Class:** mlreportgen.dom.LinkTarget

**Package:** mlreportgen.dom

Append content to link target

### Syntax

```
textObj = append(targetObj, text)
textObj = append(targetObj, text, styleName)
textObj = append(targetObj, textObj)
autoNumberObj = append(targetObj, autoNumberObj)
```

### Description

`textObj = append(targetObj, text)` converts `text` to an `mlreportgen.dom.Text` object, appends the text to the link target, and returns the text object.

`textObj = append(targetObj, text, styleName)` converts `text` to an `mlreportgen.dom.Text` object, appends the text to the link target, and returns the text object.

`textObj = append(targetObj, textObj)` appends the content of an `mlreportgen.dom.Text` object.

`autoNumberObj = append(targetObj, autoNumberObj)` appends an automatically generated number to the link target.

### Examples

#### Append Text to a Link Target

This example creates a two-page document with a link to a target at the top of the document.

Create a link target 'home' and append text to it. After a page break, create a link to the target, using `InternalLink`. The text for the link is `Go to top`.

```
import mlreportgen.dom.*
d = Document('mydoc', 'pdf');

target = LinkTarget('home');
append(target, ' - top of page');
append(d, target);

p = Paragraph('This is another paragraph');
p.Style = {PageBreakBefore(true)};
append(d, p);

append(d, InternalLink('home', 'Go to top'));
```

```
close(d);  
rptview(d.OutputPath);
```

### **Append an Automatically Generated Number to a Link Target**

This example creates a two-page document with an autonumber appended to the link target.

Create a paragraph and define an autonumber. Append the autonumber to the target and append the target to the paragraph. After the page break, create a link to the target.

```
import mlreportgen.dom.*  
d = Document('mydoc', 'docx');  
  
p = Paragraph('Chapter ');  
p.Style = {CounterInc('chapter'), WhiteSpace('preserve')};  
number = AutoNumber('chapter');  
target = LinkTarget('chapno');  
append(target, number);  
append(p, target);  
append(d, p);  
  
p = Paragraph('Paragraph on another page');  
p.Style = {PageBreakBefore(true)};  
append(d, p);  
  
append(d, InternalLink('target', 'Chapter reference'));  
  
close(d);  
rptview(d.OutputPath);
```

## **Input Arguments**

### **targetObj — Link target to append content to**

`mlreportgen.dom.LinkTarget` object

Link target to append content to, specified as an `mlreportgen.dom.LinkTarget` object.

### **text — Text to append**

character vector

Text to append, specified as a character vector.

### **styleName — Name of style**

character vector

Name of style, specified as a character vector.

### **textObj — Text object containing the text to append**

`mlreportgen.dom.Text` object

Text object containing the text to append, specified as an `mlreportgen.dom.Text` object.

### **autoNumberObj — Automatically generated number**

`mlreportgen.dom.AutoNumber` object

Automatically generated number, specified as an `m1reportgen.dom.AutoNumber` object.

## Output Arguments

### **textObj** — Text object

`m1reportgen.dom.Text` object

Text object, represented by an `m1reportgen.dom.Text` object.

### **autoNumberObj** — Automatically generated number for link target

`m1reportgen.dom.AutoNumber` object

Automatically generated number for link target, returned as an `m1reportgen.dom.AutoNumber` object.

## See Also

`m1reportgen.dom.Text` | `m1reportgen.dom.AutoNumber` | `m1reportgen.dom.LinkTarget`

## Topics

“Add Content to a Report” on page 13-10

## Introduced in R2014b

## dispatch

**Class:** mlreportgen.dom.MessageDispatcher

**Package:** mlreportgen.dom

Dispatch DOM status message

### Syntax

```
dispatch(dispatcher,message)
```

### Description

dispatch(dispatcher,message) dispatches a DOM status message.

### Examples

#### Add and Dispatch a Progress Message

This example shows how to add a progress message to display when generating a report.

Add a dispatcher and listener to the report.

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);

d.Tag = 'My report';
    dispatcher = MessageDispatcher.getTheDispatcher;
l = addlistener(dispatcher,'Message', ...
    @(src,evtdata) disp(evtdata.Message.formatAsText));

open(d);
dispatch(dispatcher,ProgressMessage('starting chapter',d));

p = Paragraph('Chapter ');
p.Tag = 'chapter title';
append(d,p);

close(d);
rptview('test',doctype);

delete (l);
```



Check the progress messages in the MATLAB Command Window. The `starting` chapter message appears, in addition to the predefined DOM progress messages.

## Input Arguments

### **dispatcher** — DOM message dispatcher

`mlreportgen.dom.MessageDispatcher` object

DOM message dispatcher, specified as an `mlreportgen.dom.MessageDispatcher` object.

### **message** — Message to dispatch

message object

Use one of the following types of DOM message objects:

- `mlreportgen.dom.ProgressMessage`
- `mlreportgen.dom.WarningMessage`
- `mlreportgen.dom.ErrorMessage`
- `mlreportgen.dom.DebugMessage`

## See Also

`mlreportgen.dom.MessageDispatcher.getTheDispatcher` |  
`mlreportgen.dom.MessageEventData` | `mlreportgen.dom.MessageFilter`

## Topics

“Display Progress and Debugger Messages” on page 13-115

## Introduced in R2014b

## mlreportgen.dom.MessageDispatcher.getTheDispatcher

**Class:** mlreportgen.dom.MessageDispatcher

**Package:** mlreportgen.dom

Return DOM message dispatcher

### Syntax

```
mlreportgen.dom.MessageDispatcher.getTheDispatcher
```

### Description

mlreportgen.dom.MessageDispatcher.getTheDispatcher returns the DOM message dispatcher. There is only one DOM message dispatcher per MATLAB session.

### Examples

#### Add a Dispatcher and Dispatch a Progress Message

This example shows how to return the DOM message dispatcher and use it to dispatch a progress message.

Add a dispatcher and listener to the report.

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);
d.Tag = 'My report';

dispatcher = MessageDispatcher.getTheDispatcher;
l = addlistener(dispatcher,'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsText));

open(d);
dispatch(dispatcher, ProgressMessage('starting chapter',d));

p = Paragraph('Chapter 1');
p.Tag = 'chapter title';
append(d, p);

close(d);
rptview('test',doctype);

delete (l);
```

Check the progress messages in the MATLAB Command Window. The `starting` chapter message appears, in addition to the predefined DOM progress messages.

**See Also**

`dispatch` | `mlreportgen.dom.MessageEventData` | `mlreportgen.dom.MessageFilter`

**Topics**

“Display Progress and Debugger Messages” on page 13-115

**Introduced in R2014b**

## toString

**Class:** `mlreportgen.dom.Number`

**Package:** `mlreportgen.dom`

Convert number to formatted text

### Syntax

```
formattedNumber = toString(numberObj)
```

### Description

`formattedNumber = toString(numberObj)` converts the number specified by `numberObj` to formatted text.

The conversion uses the first of these format specifications that it finds:

- 1 The specification in an `mlreportgen.dom.NumberFormat` object in the `Style` property of the `mlreportgen.dom.Number` object specified by `numberObj`
- 2 The specification in a `NumberFormat` object in the `Style` property of an element, such as a paragraph, list, or table, that contains the specified `Number` object
- 3 The default specification set by `mlreportgen.dom.setDefaultNumberFormat`

If the conversion does not find a format specification, the conversion uses the maximum number of digits needed to represent the number accurately.

You can use this method to see the formatted text that results from adding a `mlreportgen.dom.Number` object to a report.

### Input Arguments

**numberObj — Number to convert**

`mlreportgen.dom.Number` object

Number to convert, specified as an `mlreportgen.dom.Number` object.

### Output Arguments

**formattedNumber — Formatted number**

character vector

Formatted number, returned as a character vector.

### Examples

## Convert Number to Formatted Text

In this example, `toString` uses the format specified by a `NumberFormat` object that is included in the `Style` property of the `Number` object.

```
import mlreportgen.dom.*
numberObj = Number(pi);
numberObj.Style = [numberObj.Style {NumberFormat("%0.2f")}];
formattedNumber = toString(numberObj)
```

```
formattedNumber =
'3.14'
```

## See Also

[sprintf](#) | [mlreportgen.dom.getDefaultNumberFormat](#) | [mlreportgen.dom.Number](#) | [mlreportgen.dom.NumberFormat](#)

## Topics

“Format Numbers in a Table” on page 17-112

## Introduced in R2021a

## append

**Class:** mlreportgen.dom.OrderedList

**Package:** mlreportgen.dom

Append content to ordered list

### Syntax

```
listItemObjOut = append(orderedList,listItemObj)
listItemsOut = append(orderedList,listItems)
listObjOut = append(orderedList,list)

customElementOut = append(orderedList,customElementObj)
```

### Description

`listItemObjOut = append(orderedList,listItemObj)` appends a list item to an ordered list.

`listItemsOut = append(orderedList,listItems)` appends matrix or a cell array of list items.

`listObjOut = append(orderedList,list)` appends an ordered or unordered list.

`customElementOut = append(orderedList,customElementObj)` appends a custom element.

### Examples

#### Append Three List Items

Add three items to a list.

```
import mlreportgen.dom.*;
myReport = Document('myDoc','html');

ol = OrderedList({'Item 1' 'Item 2'});
append(myReport,ol);
append(ol,{'Item 3' 'Item 4' 'Item 5'});

close(myReport);
rptview('myDoc','html');
```

#### Append an Unordered List

```
import mlreportgen.dom.*;
myReport = Document('myDoc','html');

ol = OrderedList({'Item 1' 'Item 2'});
append(myReport,ol);
```

```
ulist = UnorderedList({'subitem1' 'subitem2'});
append(ol,ulist);

close(myReport);
rptview('myDoc','html');
```

### Append an Ordered Sublist

```
import mlreportgen.dom.*;
myReport = Document('myDoc','html');
ol = OrderedList({'a1',OrderedList({'a1','a2','b2'}),'b1'});
append(myReport,ol);
close(myReport);
rptview('myDoc','html');
```

### Append an Unordered Sublist

```
import mlreportgen.dom.*;
myReport = Document('myDoc','html');

ol = OrderedList({'a1',{'a2','b2'],'b1'});
append(myReport,ol);

close(myReport);
rptview('myDoc','html');
```

## Input Arguments

### **orderedList** – Ordered list to append content to

mlreportgen.dom.OrderedList object

Ordered list to append content to, specified as an mlreportgen.dom.OrderedList object.

### **listItemObj** – List item to append

mlreportgen.dom.ListItem object

List item to append, specified as an mlreportgen.dom.ListItem object.

### **listItems** – Items to append

matrix | cell array

A matrix can include numeric or Boolean values.

Cell array containing a combination of the following:

- A character vector
- A number
- A Boolean value
- One of the following DOM objects:

- `mlreportgen.dom.Text`
- `mlreportgen.dom.Paragraph`
- `mlreportgen.dom.ExternalLink`
- `mlreportgen.dom.InternalLink`
- `mlreportgen.dom.Table`
- `mlreportgen.dom.Image`
- `mlreportgen.dom.CustomElement`
- Horizontal one-dimensional array (for a sublist)

To append an ordered list, use an `OrderedList` DOM object instead of using the `listContent` argument.

**list — List to append**

`mlreportgen.dom.OrderedList` object | `mlreportgen.dom.UnorderedList` object

List to append, specified as an `mlreportgen.dom.OrderedList` or `mlreportgen.dom.UnorderedList` object.

**customElementObj — Custom element to append**

`mlreportgen.dom.CustomElement` object

The custom element must be a valid HTML or Word child of a list, depending on whether the output type of the document to which this element is appended is HTML or Word.

## Output Arguments

**listItemObjOut — List item appended to ordered list**

`mlreportgen.dom.ListItem` object

List items appended to an ordered list, represented by an `mlreportgen.dom.ListItem` object, matrix, or cell array. If you use the .

**listItemsOut — List items appended to ordered list**

matrix | cell array

List items appended to an ordered list, represented by a matrix or cell array, depending on the format used for the `listItems` input argument.

**listObjOut — List item appended to ordered list**

`mlreportgen.dom.ListItem` object

List items appended to an ordered list, represented by an `mlreportgen.dom.ListItem` object, matrix, or cell array. If you use the .

**customElementOut — Custom element appended to ordered list**

`mlreportgen.dom.CustomElement` object

Custom element appended to ordered list, represented by an `mlreportgen.dom.CustomElement` object.



**See Also**

`mlreportgen.dom.OrderedList` | `mlreportgen.dom.UnorderedList` |  
`mlreportgen.dom.ListItem`

**Topics**

“Create and Format Lists” on page 13-51

**Introduced in R2014b**

## append

**Class:** `mlreportgen.dom.Paragraph`

**Package:** `mlreportgen.dom`

Append content to paragraph

### Syntax

```
textObjOut = append(paraObj, text)
textObjOut = append(paraObj, text, styleName)
```

```
domObjOut = append(paraObj, domObj)
```

### Description

`textObjOut = append(paraObj, text)` creates a text object containing the specified text and appends it to a paragraph.

`textObjOut = append(paraObj, text, styleName)` creates and appends a text object using the specified style.

`domObjOut = append(paraObj, domObj)` appends a document element object, such as an image, to a paragraph.

### Examples

#### Append a Text String

```
import mlreportgen.dom.*;
d = Document('mydoc', 'html');

para = Paragraph('Results: ');
append(d, para);
append(para, 'Study 1');

close(d);
rptview(d.OutputPath);
```

#### Specify a Style for Appended Text

```
import mlreportgen.dom.*;
doc = Document('mydoc', 'docx');

para = Paragraph('Results: ', 'Title');
para.WhiteSpace = 'pre';
append(doc, para);
append(para, 'Study 2');
```

```
close(doc);
rptview('mydoc', 'docx');
```

### Append an External Link

```
import mlreportgen.dom.*;
docLink = Document('mydocLink', 'html');

mathWorksLink = ExternalLink...
    ('https://www.mathworks.com/', 'MathWorks site');
para = Paragraph('Go to the ');
append(para, mathWorksLink);
append(docLink, para);

close(docLink);
rptview(docLink.OutputPath);
```

## Input Arguments

### **paraObj** — Paragraph to append content to

`mlreportgen.dom.Paragraph` object

Paragraph to append content to, specified as an `mlreportgen.dom.Paragraph` object.

### **text** — Text to append to paragraph

character vector

Text to append to the paragraph, specified as a character vector.

### **styleName** — Name of a style to apply to text

character vector

Name of the style to define the appearance of the text. Use a style that is in the style sheet of the document that contains the paragraph.

### **domObj** — Document element to append to paragraph

DOM object

You can append the following types of document element object to a paragraph:

- `mlreportgen.dom.ExternalLink`
- `mlreportgen.dom.Image`
- `mlreportgen.dom.InternalLink`
- `mlreportgen.dom.LinkTarget`
- `mlreportgen.dom.Text`
- `mlreportgen.dom.CustomElement`

If you specify a custom element, it must be a valid HTML or Word child of the paragraph, based on the document output type.

## Output Arguments

### **textObjOut** — Text appended to paragraph

`mlreportgen.dom.Text` object

Text appended to a paragraph, represented by an `mlreportgen.dom.Text` object.

### **domObjOut** — Document element appended to paragraph

DOM object.

Document element appended to paragraph, specified as a DOM object.

## See Also

`mlreportgen.dom.Paragraph` | `clone` | `mlreportgen.dom.Text`

## Topics

“Add Content to a Report” on page 13-10

## Introduced in R2014b

# clone

**Class:** mlreportgen.dom.Paragraph

**Package:** mlreportgen.dom

Copy paragraph object

## Syntax

```
clonedPara = clone(sourcePara)
```

## Description

`clonedPara = clone(sourcePara)` copies (clones) the specified paragraph. The resulting cloned paragraph includes the children of the source paragraph, but not the parent.

## Examples

### Copy a Paragraph Object

```
import mlreportgen.dom.*;
d = Document('myDoc','html');

para1 = Paragraph('This is a paragraph');
para1.Bold = true;
append(d,para1);
para1Copy = clone(para1);
para1Copy

para1Copy =
    Paragraph with properties:

        OutlineLevel: []
            Bold: 1
            Italic: []
            Color: []
        BackgroundColor: []
            Underline: []
            WhiteSpace: []
        FontFamilyName: []
            FontSize: []
            Strike: []
            HAlign: []
        OuterLeftMargin: []
        FirstLineIndent: []
        StyleName: []
            Style: {[1x1 mlreportgen.dom.Bold]}
        CustomAttributes: []
            Parent: []
            Children: [1x1 mlreportgen.dom.Text]
                Tag: 'dom.Paragraph:2318'
                Id: '2318'
```

## Input Arguments

### sourcePara — Paragraph object to copy

`mLreportgen.dom.Paragraph` object

Paragraph object to copy, specified as an `mLreportgen.dom.Paragraph` object.

## Output Arguments

### clonedPara — Copied paragraph object

`mLreportgen.dom.Paragraph` object

Copied paragraph object, represented by an `mLreportgen.dom.Paragraph` object.

## Tips

- Use the `clone` method to append the same paragraph content more than once in a document.
- When you clone a paragraph, DOM copies all of the children objects of the source paragraph, but not the parent of the paragraph.
- The cloned paragraph includes formats that you set in the source paragraph. The cloned paragraph formats use the same format objects as the source paragraph. If you change the format setting in the shared format object, the source and cloned paragraphs reflect that change.

If you change a format setting in the cloned paragraph, then DOM creates a new format object for the cloned paragraph, using the new format setting. For that format, the source and cloned paragraph no longer share the same format object.

This example shows the relationship between the formats for the source and cloned paragraphs.

- 1 Create a paragraph that uses a style that sets the **Bold** and **Italic** formats to `true`.

```
import mLreportgen.dom.*;
myReport = Document('myDoc','html');
p = Paragraph('This is a paragraph');
append(myReport,p);
MyStyle = {Bold,Italic};
p.Style = MyStyle;
p.Bold
```

```
ans =
```

```
1
```

```
p.Italic
```

```
ans =
```

```
1
```

- 2 Clone the paragraph. The **Bold** and **Italic** formats are the same as those of the source paragraph.

```
pClone = clone(p);
pClone.Bold
```

```
ans =
  1
p.Italic
```

```
ans =
  1
```

- 3** For the cloned paragraph, change turn off bold text. The change to the **Bold** format in the cloned paragraph does not affect the text for the source paragraph. The source paragraph text is still bold.

```
pClone.Bold = false;
p.Bold
```

```
ans =
  1
```

- 4** In the style object (`MyStyle`) for the source paragraph, turn off italics. Now the cloned paragraph does not use italics, because it shares the `MyStyle` setting for the `Italics` format.

```
MyStyle(2).Value = false
pClone.Italic
```

```
ans =
  0
```

## See Also

`mreportgen.dom.Paragraph | append | mreportgen.dom.Document`

## Topics

“Add Content to a Report” on page 13-10

## Introduced in R2014b

## formatAsHTML

**Class:** `mlreportgen.dom.ProgressMessage`

**Package:** `mlreportgen.dom`

Wrap message in HTML tags

### Syntax

```
htmlMessageOut = formatAsHTML(message)
```

### Description

`htmlMessageOut = formatAsHTML(message)` returns the message text formatted with HTML tags.

### Examples

#### Format a Message as HTML

This example uses `formatAsHTML` with the Web command to display the progress messages.

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);
d.Tag = 'My report';

dispatcher = MessageDispatcher.getTheDispatcher;
l = addlistener(dispatcher,'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsHTML));

open(d);
dispatch(dispatcher, ProgressMessage('starting chapter',d));
p = Paragraph('Chapter ');
p.Tag = 'chapter title';
p.Style = { CounterInc('chapter'),...
    CounterReset('table'),WhiteSpace('pre') };
append(p,AutoNumber('chapter'));
append(d,p);

close(d);
rptview('test',doctype);

delete (l);
```

### Input Arguments

**message** — Progress message

`mlreportgen.dom.ProgressMessage` object

Progress message, specified as an `mlreportgen.dom.ProgressMessage` object.



## Output Arguments

### **htmlMessageOut — Progress message with HTML tagging**

`mlreportgen.dom.ProgressMessage` object

Progress message with HTML tagging, specified as an `mlreportgen.dom.ProgressMessage` object.

### **See Also**

[formatAsText](#) | [mlreportgen.dom.ProgressMessage](#) | [mlreportgen.dom.MessageFilter](#)

### **Topics**

“Display Progress and Debugger Messages” on page 13-115

**Introduced in R2014b**

## formatAsText

**Class:** mlreportgen.dom.ProgressMessage

**Package:** mlreportgen.dom

Format message as text

### Syntax

```
textMessageOut = formatAsText(message)
```

### Description

`textMessageOut = formatAsText(message)` returns the message text formatted as text.

### Examples

#### Format a Message with White Spaces

This example uses `formatAsText` with the Web command to display the progress messages.

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);
d.Tag = 'My report';

dispatcher = MessageDispatcher.getTheDispatcher;
l = addlistener(dispatcher,'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsText));

open(d);
dispatch(dispatcher,ProgressMessage('starting chapter',d));
p = Paragraph('Chapter ');
p.Tag = 'chapter title';
p.Style = { CounterInc('chapter'),...
    CounterReset('table'),WhiteSpace('pre') };
append(p, AutoNumber('chapter'));
append(d,p);

close(d);
rptview('test',doctype);

delete(l);
```

### Input Arguments

**message** — The DOM progress message

mlreportgen.dom.ProgressMessage object

The DOM message, specified as an mlreportgen.dom.ProgressMessage object.

## Output Arguments

### **textMessageOut — DOM progress message formatted as text**

`mlreportgen.dom.ProgressMessage` object

DOM progress message formatted as text, represented by an `mlreportgen.dom.ProgressMessage` object.

## See Also

[formatAsHTML](#) | [mlreportgen.dom.ProgressMessage](#) | [mlreportgen.dom.MessageFilter](#)

## Topics

“Display Progress and Debugger Messages” on page 13-115

**Introduced in R2014b**

## passesFilter

**Class:** mlreportgen.dom.ProgressMessage

**Package:** mlreportgen.dom

Determine if message passes filter

### Syntax

```
tf = passesFilter(message,filter)
```

### Description

tf = passesFilter(message,filter) determines whether the message passes the filter.

### Examples

#### Determine Whether a Message Passes a Filter

This example shows how to add a progress message to display when generating a report.

Add a dispatcher and listener to the report. Configure the dispatcher to include debug messages.

```
import mlreportgen.dom.*;
d = Document('test','html');

dispatcher = MessageDispatcher.getTheDispatcher;
dispatcher.Filter.DebugMessagesPass = true;
l = addlistener(dispatcher,'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsText));
```

Create a progress message.

```
open(d);
dispatch(dispatcher, ProgressMessage('starting chapter',d));
p = Paragraph('Chapter ');
p.Tag = 'chapter title';
p.Style = { CounterInc('chapter'),...
    CounterReset('table'),WhiteSpace('pre') };
append(p,AutoNumber('chapter'));
append(d,p);
```

Generate the report and delete the listener.

```
close(d);
rptview('test','html');

delete(l);
```

Check the progress messages in the MATLAB Command Window. In addition to the predefined DOM progress messages, the `starting` chapter message added in this example appears. The output also includes debug messages.

## Input Arguments

### **message** — DOM progress message

`mlreportgen.dom.ProgressMessage` object

DOM progress message, specified as an `mlreportgen.dom.ProgressMessage` object.

### **filter** — Filter to use with message

`mlreportgen.dom.MessageFilter` object

Filter to use with the progress message, specified as an `mlreportgen.dom.MessageFilter` object.

## Output Arguments

### **tf** — Indication of whether the message passes the filter

a `Boolean`

- 1 — Messages passes the specified filter (the dispatcher handles the message)
- 0 — Messages fails the specified filter (the dispatcher ignores the message)

## See Also

`mlreportgen.dom.MessageFilter` | `mlreportgen.dom.ProgressMessage`

## Topics

“Display Progress and Debugger Messages” on page 13-115

**Introduced in R2014b**

## entry

**Class:** mlreportgen.dom.Table

**Package:** mlreportgen.dom

Access table entry

### Syntax

```
tableEntryOut = entry(tableObj,row,column)
```

### Description

`tableEntryOut = entry(tableObj, row, column)` returns the table entry for the specified column of the specified row.

### Examples

#### Color a Table Entry

Color the table entry in row 3, column 4.

```
import mlreportgen.dom.*;
rpt = Document("myDoc","html");
t = Table(magic(5));
t.Border = "single";
t.ColSep = "single";
t.RowSep = "single";
t.TableEntriesInnerMargin = "2pt";
t.TableEntriesHAlign = "right";
entry34 = entry(t,3,4);
entry34.Children(1).Style = [entry34.Children(1).Style {Color("red")}];
append(rpt,t);
close(rpt);
rptview(rpt);
```

### Input Arguments

#### **tableObj** — Table containing the entry

mlreportgen.dom.Table object | mlreportgen.dom.FormalTable object

#### **row** — Table row containing the entry

number

Index number of the row (top row is row 1).

Data Types: double

#### **column** — Column containing the entry

number

Index number of the column (in a left-to-right text flow table, the left-most column is 1).

Data Types: double

## **Output Arguments**

**tableEntryOut** — Table entry object

mlreportgen.dom.TableEntry object

## **See Also**

mlreportgen.dom.TableEntry | row

## **Topics**

“Create Informal Tables” on page 13-64

**Introduced in R2014b**

## row

**Class:** mlreportgen.dom.Table

**Package:** mlreportgen.dom

Access table row

### Syntax

```
tableRowOut = row(tableObj,row)
```

### Description

`tableRowOut = row(tableObj,row)` returns the table row at the specified row number.

### Examples

#### Color a Table Row

Color the second row of a table.

```
import mlreportgen.dom.*;
myReport = Document('myDoc','html');

t = Table(magic(5));
te = row(t,2);
te.Style = {Color('red')};
append(myReport,t);

close(myReport);
rptview(myReport.OutputPath);
```

### Input Arguments

#### **tableObj** – Table containing entry

mlreportgen.dom.Table object

Table containing the entry, specified as an mlreportgen.dom.Table object.

#### **row** – Table row

number

Index number of the row (top row is row 1).

Data Types: double

### Output Arguments

#### **tableRowOut** – Table row object

mlreportgen.dom.TableRow object



Table row object, represented by an `mlreportgen.dom.TableRow` object.

**See Also**

`mlreportgen.dom.TableRow` | `mlreportgen.dom.TableEntry`

**Topics**

“Create Informal Tables” on page 13-64

**Introduced in R2014b**

## append

**Class:** mlreportgen.dom.TableRow

**Package:** mlreportgen.dom

Append content to table row

### Syntax

```
entryOut = append(rowObj,entryObj)
```

### Description

`entryOut = append(rowObj,entryObj)` appends an entry to a table row.

### Examples

#### Append a Table Entry to a Row

Create a two-column table.

```
import mlreportgen.dom.*;
myReport = Document('myDoc','html');
table = Table(2);
table.Style = {Border('solid'),RowSep('solid'),ColSep('solid')};
table.TableEntriesStyle = {Width('2in'),HAlign('center')};
```

Create three table rows with entries. Append each entry to a row using `append(row,te)`.

```
for i=1:3
    row = TableRow();
    te = TableEntry();
    append(te,Text([num2str(i) ' - 1']));
    append(row,te);
    te = TableEntry();
    append(te,Text([num2str(i) ' - 2']));
    append(row,te);
    append(table,row);
end
```

Append the table and display the report.

```
append(myReport,table);

close(myReport);
rptview(myReport.OutputPath);
```

### Input Arguments

**rowObj** — Row to append the table entry to  
mlreportgen.dom.TableRow object

Row to append the table entry to, specified as an `mlreportgen.dom.TableRow` object.

**entryObj – Table entry to append**

`mlreportgen.dom.TableEntry` object

Table entry to append, specified as an `mlreportgen.dom.TableEntry` object.

## Output Arguments

**entryOut – Appended table entry**

`mlreportgen.dom.TableEntry` object

Appended table entry, represented by an `mlreportgen.dom.TableEntry` object.

## See Also

`mlreportgen.dom.TableRow` | `mlreportgen.dom.TableEntry`

## Topics

“Create Informal Tables” on page 13-64

**Introduced in R2014b**

## add

**Class:** mlreportgen.ppt.ContentPlaceholder

**Package:** mlreportgen.ppt

Add paragraphs to content placeholder

### Syntax

```
addedParagraphObj = add(contentPlaceholder,paragraph)
add(contentPlaceholder,paragraphs)
```

### Description

addedParagraphObj = add(contentPlaceholder,paragraph) adds one paragraph to a content placeholder and returns the paragraph object.

add(contentPlaceholder,paragraphs) adds multiple paragraphs to a content placeholder.

### Examples

#### Add Paragraph to Content Placeholder

To add text to a content placeholder in a slide, use the add method of the ContentPlaceholder object.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation and add a Title and Content slide to the presentation.

```
ppt = Presentation("myContentPlaceholderAddPresentation.pptx");
open(ppt);
slide = add(ppt,"Title and Content");
```

Find the text box placeholder for the title and replace the content.

```
titlePlaceholderObj = find(slide,"Title");
replace(titlePlaceholderObj,"Fruits and Vegetables");
```

Find the content placeholder and replace the content with a paragraph.

```
contents = find(slide,"Content");
contentPlaceholderObj = contents(1);
replace(contentPlaceholderObj,{'Fruits' ["Apples" "Oranges"]});
```

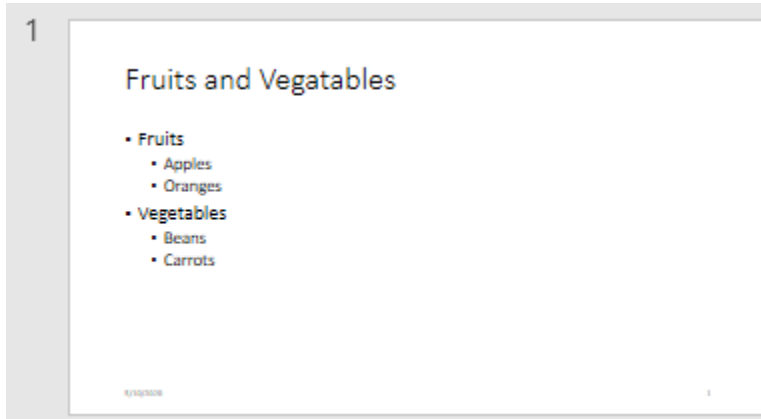
Add another paragraph to the content placeholder.

```
add(contentPlaceholderObj,{"Vegetables" ["Beans" "Carrots"]});
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here is the slide in the generated presentation:



## Input Arguments

### **contentPlaceholder** — Content placeholder

`mlreportgen.ppt.ContentPlaceholder` object

Content placeholder, specified as an `mlreportgen.ppt.ContentPlaceholder` object.

### **paragraph** — Paragraph to add to content placeholder

character vector | string scalar | `mlreportgen.ppt.Paragraph` object

Paragraph to add to the content placeholder, specified as a character vector, string scalar, or an `mlreportgen.ppt.Paragraph` object.

### **paragraphs** — Multiple paragraphs to add to content placeholder

string array | cell array of any combination of string scalars, character vectors, or `mlreportgen.ppt.Paragraph` objects

Multiple paragraphs to add to the content placeholder, specified as one of these values:

- String array
- Cell array of any combination of string scalars, character vectors, or `mlreportgen.ppt.Paragraph` objects. Inner arrays are indented from outer arrays.

The slide layout determines whether the text displays as paragraphs, bullet list items, or numbered list items.

## Output Arguments

### **addedParagraphObj** — Paragraph added to content placeholder

`mlreportgen.ppt.Paragraph` object

Paragraph added to the content placeholder, returned as an `mlreportgen.ppt.Paragraph` object.

## **See Also**

`mlreportgen.ppt.ContentPlaceholder` | `mlreportgen.ppt.Paragraph` | `replace`

## **Topics**

“Access PowerPoint Template Elements” on page 14-32

“Add and Replace Presentation Content” on page 14-58

**Introduced in R2015b**

# replace

**Class:** mlreportgen.ppt.ContentPlaceholder

**Package:** mlreportgen.ppt

Replace content placeholder or content

## Syntax

```
replacementContentObj = replace(contentPlaceholder,content)
replace(contentPlaceholder,paragraphs)
```

## Description

`replacementContentObj = replace(contentPlaceholder,content)` replaces the content of the `ContentPlaceholder` object or replaces the `ContentPlaceholder` object. If the `content` argument is a paragraph, the `replace` method replaces the content of the `ContentPlaceholder` by making the paragraph a child of the `ContentPlaceholder` object. If the `content` argument is a picture or a table, the `replace` method replaces the `ContentPlaceholder` object with an `mlreportgen.ppt.Picture` or `mlreportgen.ppt.Table` object, respectively.

`replace(contentPlaceholder,paragraphs)` replaces the content of a content placeholder with multiple paragraphs.

## Examples

### Replace Content Placeholder with Content

Add four `Title` and `Content` slides to a presentation. Replace the content placeholder of the first slide with text, the second slide with a table, the third slide with a picture, and the fourth slide with a multilevel list.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation.

```
ppt = Presentation("myContentPlaceholderPresentation.pptx");
open(ppt);
```

Add a slide with a `Title` and `Content` layout.

```
slide1 = add(ppt,"Title and Content");
```

Use the `find` method of the slide object to find the placeholder object that has the name `Title`.

```
titlePlaceholderObj1 = find(slide1,"Title");
```

The `find` method returns an `mlreportgen.ppt.TextBoxPlaceholder` object.

Replace the placeholder content with the title text.

```
replace(titlePlaceholderObj1, "Content Replaced with Paragraph");
```

Use the find method of the slide object to find the placeholder object that has the name Content.

```
contentPlaceholderObj1 = find(slide1, "Content");
```

The find method returns an `mreportgen.ppt.ContentPlaceholder` object.

Replace the placeholder content with a paragraph.

```
replace(contentPlaceholderObj1, Paragraph("This is my paragaph"));
```

Add a second slide with a Title and Content layout.

```
slide2 = add(ppt, "Title and Content");
```

Replace the placeholder for the title with the title text.

```
titlePlaceholderObj2 = find(slide2, "Title");  
replace(titlePlaceholderObj2, "Content Replaced with Table for Order 2 Magic Square");
```

Replace the content placeholder with a table.

```
contentPlaceholderObj2 = find(slide2, "Content");  
replace(contentPlaceholderObj2, Table(magic(2)));
```

Add a third slide with a Title and Content layout.

```
slide3 = add(ppt, "Title and Content");
```

Replace the placeholder for the title with the title text.

```
titlePlaceholderObj3 = find(slide3, "Title");  
replace(titlePlaceholderObj3, "Content Replaced with Picture of Peppers");
```

Replace the content placeholder with a picture.

```
contentPlaceholderObj3 = find(slide3, "Content");  
replace(contentPlaceholderObj3, Picture("peppers.png"));
```

Add a fourth slide with a Title and Content layout.

```
slide4 = add(ppt, "Title and Content");
```

Replace the placeholder for the title with the title text.

```
titlePlaceholderObj4 = find(slide4, "Title");  
replace(titlePlaceholderObj4, "Content Replaced with a Multilevel List");
```

Create content for a multilevel list. You can represent a multilevel list as a cell array that contains one or more cell arrays that represent sublists. Use an `mreportgen.ppt.Paragraph` object to format an item in the list.

```
greenTea = Paragraph("Green Tea");  
greenTea.FontColor = "green";
```

```
multilevelContent = { ...
```



```
"Coffee", ...
"Tea", ...
{ ...
  "Black Tea", ...
  greenTea, ...
}, ...
"Milk", ...
};
```

Replace the Content placeholder with the multilevel list content.

```
contentPlaceholderObj4 = find(slide4, "Content");
replace(contentPlaceholderObj4, multilevelContent);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here are the generated slides:

1

### Content Replaced with Paragraph

- This is my paragraph

11/16/2020

1

2

### Content Replaced with Table for Order 2 Magic Square

3	3
4	2

11/16/2020

2

3

### Content Replaced with Picture of Peppers



11/16/2020

3

4

### Content Replaced with a Multilevel List

- Coffee
- Tea
  - Black Tea
  - Green Tea
- Milk

11/16/2020

4

## Input Arguments

### **contentPlaceholder — Content placeholder**

`mlreportgen.ppt.ContentPlaceholder` object

Content placeholder, specified as an `mlreportgen.ppt.ContentPlaceholder` object.

### **content — Content to use as replacement**

character vector | string scalar | `mlreportgen.ppt.Paragraph` object |  
`mlreportgen.ppt.Table` object | `mlreportgen.ppt.Picture` object

Content to use as replacement, specified as a character vector, string scalar or one of these objects:

- `mlreportgen.ppt.Paragraph`
- `mlreportgen.ppt.Picture`
- `mlreportgen.ppt.Table`

### **paragraphs — Multiple paragraphs to use as replacement**

string array | cell array of any combination of string scalars, character vectors, or  
`mlreportgen.ppt.Paragraph` objects

Multiple paragraphs to use as replacement, specified as one of these values:

- String array
- Cell array of any combination of string scalars, character vectors, or  
`mlreportgen.ppt.Paragraph` objects. Inner arrays are indented from outer arrays.

The slide layout determines whether the text displays as paragraphs, bullet list items, or numbered list items.

## Output Arguments

### **replacementContentObj — Replacement content**

`mlreportgen.ppt.Paragraph` object | `mlreportgen.ppt.Table` object |  
`mlreportgen.ppt.Picture` object

Replacement content, returned as an `mlreportgen.ppt.Paragraph`,  
`mlreportgen.ppt.Picture`, or `mlreportgen.ppt.Table` object.

## See Also

`mlreportgen.ppt.ContentPlaceholder` | `add` | `mlreportgen.ppt.Paragraph` |  
`mlreportgen.ppt.Picture` | `mlreportgen.ppt.Table`

### Topics

“Access PowerPoint Template Elements” on page 14-32

“Add and Replace Presentation Content” on page 14-58

### Introduced in R2015b

## dispatch

**Class:** mlreportgen.ppt.MessageDispatcher

**Package:** mlreportgen.ppt

Dispatch PPT status message

### Syntax

```
dispatch(dispatcher,message)
```

### Description

dispatch(dispatcher,message) dispatches a PPT status message.

### Examples

#### Add and Dispatch a Progress Message

This example shows how to add a progress message to display when generating a report.

Add a dispatcher and listener to the presentation.

```
import mlreportgen.ppt.*;
pre = Presentation('myPresentation.pptx');

dispatcher = MessageDispatcher.getTheDispatcher;
l = addlistener(dispatcher,'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsText));

dispatch(dispatcher,ErrorMessage('invalid slide',pre));
open(pre);

titleText = Text('This is a Title');
titleText.Style = {Bold};

replace(pre,'Title',titleText);

close(pre);

delete(l);
```

Check the progress messages in the MATLAB Command Window. The starting chapter message appears, in addition to the predefined PPT progress messages.

### Input Arguments

#### dispatcher — PPT message dispatcher

mlreportgen.ppt.MessageDispatcher object

PPT message dispatcher, specified as an mlreportgen.ppt.MessageDispatcher object.

**message — Message to dispatch**

PPT message object

Message to dispatch, specified as a PPT message object. Use one of these types of PPT message objects:

- `mlreportgen.ppt.ProgressMessage`
- `mlreportgen.ppt.WarningMessage`
- `mlreportgen.ppt.ErrorMessage`
- `mlreportgen.ppt.DebugMessage`

**See Also**

`getTheDispatcher` | `mlreportgen.ppt.MessageEventData` |  
`mlreportgen.ppt.MessageFilter`

**Topics**

“Display Presentation Generation Messages” on page 14-14

**Introduced in R2015b**

## mlreportgen.ppt.MessageDispatcher.getTheDispatcher

**Class:** mlreportgen.ppt.MessageDispatcher

**Package:** mlreportgen.ppt

Return PPT message dispatcher

### Syntax

```
mlreportgen.ppt.MessageDispatcher.getTheDispatcher
```

### Description

`mlreportgen.ppt.MessageDispatcher.getTheDispatcher` returns the PPT message dispatcher. There is only one PPT message dispatcher per MATLAB session.

### Examples

#### Add a Dispatcher and Dispatch a Progress Message

This example shows how to return the PPT message dispatcher and use it to dispatch a progress message.

Add a dispatcher and listener to the report.

```
import mlreportgen.ppt.*;
pre = Presentation('myPresentation.pptx');

dispatcher = MessageDispatcher.getTheDispatcher;
l = addlistener(dispatcher, 'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsText));

dispatch(dispatcher, ProgressMessage...
    ('Empty presentation will be created', pre));

open(pre);
close(pre);

delete(l);
```

Check the progress messages in the MATLAB Command Window. The starting chapter message appears, in addition to the predefined PPT progress messages.

### See Also

[dispatch](#) | [mlreportgen.ppt.MessageEventData](#) | [mlreportgen.ppt.MessageFilter](#)

### Topics

“Display Presentation Generation Messages” on page 14-14

**Introduced in R2015b**

## append

**Class:** mlreportgen.ppt.Paragraph

**Package:** mlreportgen.ppt

Append content to paragraph

### Syntax

```
contentObj = append(paragraph, content)
```

### Description

`contentObj = append(paragraph, content)` appends content to a paragraph.

### Examples

#### Append Formatted Text and External Link to Paragraph

Create a presentation.

```
import mlreportgen.ppt.*  
  
ppt = Presentation('myParagraphPresentation.pptx');  
open(ppt);  
add(ppt, 'Title Slide');  
add(ppt, 'Title and Content');
```

Create a Paragraph object to use for the title of slides. Make the text bold and red.

```
p1 = Paragraph('Title for ');
```

Add more text to the title.

```
text = append(p1, 'My Presentation');  
text.Bold = true;  
text.FontColor = 'red';
```

Replace the title with the p1 paragraph.

```
replace(ppt, 'Title', p1);
```

Create a paragraph for the content of the second slide.

```
p2 = Paragraph('Click the link for the ');  
contentObj = append(p2, ExternalLink('https://www.mathworks.com', 'MathWorks site.'));
```

Replace the content with the p2 paragraph.

```
replace(ppt, 'Content', p2);
```

Close and view the presentation.



```
close(ppt);  
rptview(ppt);
```

## Input Arguments

### **paragraph** — Paragraph to append content to

`mlreportgen.ppt.Paragraph` object

Paragraph to append content to, specified as an `mlreportgen.ppt.Paragraph` object.

### **content** — Content to append to paragraph

character vector | `mlreportgen.ppt.Text` object | `mlreportgen.ppt.InternalLink` | `mlreportgen.ppt.ExternalLink` object

Content to add to a paragraph, specified as a character vector, or an `mlreportgen.ppt.Text`, `mlreportgen.ppt.InternalLink`, or `mlreportgen.ppt.ExternalLink` or an object.

## Output Arguments

### **contentObj** — Content object

`mlreportgen.ppt.Text` object | `mlreportgen.ppt.InternalLink` | `mlreportgen.ppt.ExternalLink` object

Content object, returned as an `mlreportgen.ppt.Text`, `mlreportgen.ppt.InternalLink`, or `mlreportgen.ppt.ExternalLink` or an object.

## See Also

`mlreportgen.ppt.Paragraph` | `mlreportgen.ppt.Text` | `mlreportgen.ppt.InternalLink` | `mlreportgen.ppt.ExternalLink`

### Topics

“Add and Replace Text” on page 14-59

“Create and Format Paragraphs” on page 14-67

### Introduced in R2015b

## clone

**Class:** `mlreportgen.ppt.Paragraph`

**Package:** `mlreportgen.ppt`

Copy PPT API paragraph

### Syntax

```
objCopy = clone(obj)
```

### Description

`objCopy = clone(obj)` returns a copy of the `mlreportgen.ppt.Paragraph` object specified by `obj`. The copy includes the children of the source paragraph, but not the parent.

### Input Arguments

**obj** — Paragraph object to clone

`mlreportgen.ppt.Paragraph` object

Paragraph object to clone, specified as an `mlreportgen.ppt.Paragraph` object.

### Output Arguments

**objCopy** — Paragraph object copy

`mlreportgen.ppt.Paragraph` object

Paragraph object copy, returned as an `mlreportgen.ppt.Paragraph` object.

## Examples

### Reuse a Paragraph in a Presentation

In this example, you use the same paragraph content in two slides by making a copy of the paragraph used in the first slide and appending the copy to the second slide.

Import the PPT API package so that you do not have to use long, fully qualified class names.

```
import mlreportgen.ppt.*;
```

Create and open a presentation. Add a slide to the presentation.

```
ppt = Presentation("myPresentation.pptx");  
open(ppt);  
slide1 = add(ppt,"Title and Content");
```

Create a Paragraph object and set the Bold and Italic properties to true. Replace the content of `slide1` with the Paragraph object.

```
p = Paragraph("My slide content");
p.Bold = true;
p.Italic = true;
replace(slide1, "Content", p);
```

Add a second slide to the presentation.

```
slide2 = add(ppt, "Title and Content");
```

Make a copy of the `Paragraph` object that you added to `slide1`. The attribute values of the copy have the same values as the source `Paragraph` object.

```
pcopy = clone(p)
```

```
pcopy =
```

```
  Paragraph with properties:
```

```

        Bold: 1
        Font: []
ComplexScriptFont: []
        FontColor: []
        FontSize: []
        Italic: 1
        Strike: []
        Subscript: []
        Superscript: []
        Underline: []
        HAlign: []
        Level: []
        Style: {[1x1 mlreportgen.ppt.Bold] [1x1 mlreportgen.ppt.Italic]}
        Children: [1x1 mlreportgen.ppt.Text]
        Parent: []
        Tag: 'ppt.Paragraph:66'
        Id: '66'
```

Append additional content to the copy of the `Paragraph` object.

```
append(pcopy, " for the second slide");
```

Replace the content of the second slide with the copy.

```
replace(slide2, "Content", pcopy);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

### Clone Paragraph and Modify Formats

This example shows the relationship between the formats of a paragraph and a copy of the paragraph.

When you set a format property in a `mlreportgen.ppt.Paragraph` object, the PPT API adds a corresponding format object to the `Style` property of the `Paragraph` object. When you clone a

Paragraph object, the copy has the same format property values and includes the same format objects as the source Paragraph object. If you change a format setting in a shared format object, the source Paragraph object and the copy reflect the change. If you change a format property of the source Paragraph object or the copy, the PPT API adds a new format object to the Style property of the object that you changed. Because the source Paragraph object and the copy no longer share the same format object, the change to the format property in one of the Paragraph objects does not affect the other object.

Create an `mlreportgen.ppt.Paragraph` object with the `Bold` and `Italic` properties set to `true`.

```
import mlreportgen.ppt.*;
p = Paragraph("My slide content");
p.Bold = true;
p.Italic = true;
```

The PPT API sets the `Style` property of the Paragraph object to a cell array that contains an `mlreportgen.ppt.Bold` object with `Value` set to `true` and an `mlreportgen.ppt.Italic` object with `Value` set to `true`.

`p.Style`

```
ans=1x2 cell array
    {1x1 mlreportgen.ppt.Bold}    {1x1 mlreportgen.ppt.Italic}
```

Clone the Paragraph object. The `Bold`, `Italic`, and `Style` property values are the same as those of the source Paragraph object.

```
pClone = clone(p);
pClone.Bold
```

```
ans = logical
     1
```

`pClone.Italic`

```
ans = logical
     1
```

`pClone.Style`

```
ans=1x2 cell array
    {1x1 mlreportgen.ppt.Bold}    {1x1 mlreportgen.ppt.Italic}
```

In the `Style` property of the source Paragraph object, set the `Value` property of the `Italic` object to `false`. This change affects the copy of the Paragraph object because its `Style` property includes the same `Italic` object as the source Paragraph object. The PPT API sets the value of the `Italic` property of the Paragraph copy to `false` so that it matches the value of the `Italic` object included in the `Style` property.

```
p.Style{2}.Value = false;
pClone.Italic
```

```
ans = logical
     0
```

In the copy of the Paragraph object, set the `Bold` property to `false`. This change does not affect the source Paragraph object. In the source Paragraph object, the `Bold` property is still `true`.

```
pClone.Bold = false;  
p.Bold
```

```
ans = logical  
    1
```

## Tips

- Use the `clone` method to include the same paragraph content more than once in a presentation.
- When you clone a paragraph, the PPT API copies all of the children objects of the source paragraph, but not the parent of the paragraph.
- The paragraph copy includes formats that you set in the source paragraph. The formats of the paragraph copy use the same format objects as the source paragraph. If you change the format setting in the shared format object, the source and copy reflect the change.

If you change a format setting in the paragraph copy, the PPT API creates a new format object for the copy, using the new format setting. For that format, the source paragraph and the copy no longer share the same format object.

## See Also

`mlreportgen.ppt.Paragraph` | `mlreportgen.ppt.Presentation` | `append`

## Topics

“Create and Format Paragraphs” on page 14-67

“Presentation Formatting Approaches” on page 14-18

## Introduced in R2021b

## replace

**Class:** mlreportgen.ppt.Picture

**Package:** mlreportgen.ppt

Replace picture

### Syntax

```
pictureObj = replace(picture, replacementPicture)
```

### Description

`pictureObj = replace(picture, replacementPicture)` replaces a picture with another picture.

### Examples

#### Replace a Picture

Create a presentation.

```
import mlreportgen.ppt.*  
  
ppt = Presentation('myPictureReplacePresentation');  
slide1 = add(ppt, 'Blank');
```

Create an mlreportgen.ppt.Picture object.

```
plane = Picture(which('b747.jpg'));  
plane.X = '1in';  
plane.Y = '1in';  
plane.Width = '5in';  
plane.Height = '2in';
```

Add the picture to the slide.

```
add(slide1, plane);
```

Create a second picture.

```
peppers = Picture(which('peppers.png'));  
peppers.X = '1in';  
peppers.Y = '1in';  
peppers.Width = '3in';  
peppers.Height = '3in';
```

Replace the plane picture with the peppers picture.

```
replace(plane, peppers);
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

## Input Arguments

### **picture** — Picture to replace

mlreportgen.ppt.Picture object

Picture to replace, specified as an mlreportgen.ppt.Picture object.

### **replacementPicture** — Picture to use as replacement

mlreportgen.ppt.Picture object

Picture to use as a replacement, specified as an mlreportgen.ppt.Picture object.

## Output Arguments

### **pictureObj** — Picture

mlreportgen.ppt.Picture object

Picture, returned as an mlreportgen.ppt.Picture object.

## See Also

mlreportgen.ppt.Picture

### Topics

“Access PowerPoint Template Elements” on page 14-32

“Add or Replace a Picture” on page 14-62

### Introduced in R2015b

## replace

**Class:** mlreportgen.ppt.PicturePlaceholder

**Package:** mlreportgen.ppt

Replace picture placeholder with picture

### Syntax

```
replacementPictureObj = replace(picturePlaceholder,picture)
```

### Description

`replacementPictureObj = replace(picturePlaceholder,picture)` replaces the specified picture placeholder with the specified picture and returns the picture object.

### Examples

#### Replace Picture Placeholder with Picture

Add a `Title` and `Picture` slide to a presentation and then replace the title and picture placeholders with your own title and picture.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation.

```
ppt = Presentation("myPicturePlaceholderPresentation.pptx");  
open(ppt);
```

Add a slide that has a `Title` and `Picture` layout.

```
slide = add(ppt,"Title and Picture");
```

Use the `find` method of the slide object to find the placeholder object that has the name `Title`.

```
titlePlaceholderObj = find(slide,"Title")
```

```
titlePlaceholderObj =  
    TextBoxPlaceholder with properties:
```

```
        Bold: []  
        Font: []  
ComplexScriptFont: []  
        FontColor: []  
        FontSize: []  
        Italic: []  
        Strike: []  
        Subscript: []
```



```

    Superscript: []
    Underline: []
    BackgroundColor: []
    VAlign: []
    Name: 'Title'
    X: []
    Y: []
    Width: []
    Height: []
    Style: []
    Children: []
    Parent: [1x1 mlreportgen.ppt.Slide]
    Tag: 'ppt.TextBoxPlaceholder:6:9'
    Id: '6:9'

```

The `find` method returns an `mlreportgen.ppt.TextBoxPlaceholder` object.

Replace the placeholder content with the title text.

```
replace(titlePlaceholderObj, "Airplane");
```

Use the `find` method of the slide object to find the placeholder object that has the name `Picture`.

```
picturePlaceholderObj = find(slide, "Picture")
```

```
picturePlaceholderObj =
    PicturePlaceholder with properties:
```

```

    Bold: []
    Font: []
    ComplexScriptFont: []
    FontColor: []
    FontSize: []
    Italic: []
    Strike: []
    Subscript: []
    Superscript: []
    Underline: []
    BackgroundColor: []
    VAlign: []
    Name: 'Picture'
    X: []
    Y: []
    Width: []
    Height: []
    Style: []
    Children: []
    Parent: [1x1 mlreportgen.ppt.Slide]
    Tag: 'ppt.PicturePlaceholder:7:10'
    Id: '7:10'

```

The `find` method returns an `mlreportgen.ppt.PicturePlaceholder` object.

Replace the picture placeholder with a picture.

```
replace(picturePlaceholderObj, Picture("b747.jpg"));
```

Close and view the presentation

```
close(ppt);  
rptview(ppt);
```

## Input Arguments

### **picturePlaceholder** — Picture placeholder

`mlreportgen.ppt.PicturePlaceholder` object

Picture placeholder, specified as an `mlreportgen.ppt.PicturePlaceholder` object.

### **picture** — Picture to use as replacement

`mlreportgen.ppt.Picture` object

Picture to use as the replacement for the picture placeholder, specified as an `mlreportgen.ppt.Picture` object.

## Output Arguments

### **replacementPictureObj** — Replacement picture

`mlreportgen.ppt.Picture` object

Replacement picture, returned as an `mlreportgen.ppt.Picture` object.

## See Also

`mlreportgen.ppt.PicturePlaceholder` | `mlreportgen.ppt.Picture`

### Topics

“Access PowerPoint Template Elements” on page 14-32

“Add or Replace a Picture” on page 14-62

### Introduced in R2015b

# add

**Class:** mlreportgen.ppt.Presentation

**Package:** mlreportgen.ppt

Add slide to presentation

## Syntax

```
slideObj = add(presentation, slideLayout)
slideObj = add(presentation, slideLayout, slideMaster)
slideObj = add(presentation, slideLayout, otherSlide)
slideObj = add(presentation, slideLayout, slideMaster, otherSlide)
slideObj = add(presentation, slideLayout, index)
slideObj = add(presentation, slideLayout, slideMaster, index)
```

## Description

`slideObj = add(presentation, slideLayout)` adds a slide to the presentation using the specified slide layout name. If the specified layout occurs in multiple slide masters, the PPT API uses the first occurrence of the layout that it finds in the template.

`slideObj = add(presentation, slideLayout, slideMaster)` uses the specified slide layout in the specified slide master. Use the `slideMaster` argument when the specified layout occurs in multiple slide masters.

`slideObj = add(presentation, slideLayout, otherSlide)` adds the slide immediately before the slide specified in the `otherSlide` argument, using the specified slide layout in the first slide master in the presentation.

`slideObj = add(presentation, slideLayout, slideMaster, otherSlide)` adds the slide immediately before the `otherSlide` slide, using the specified slide layout in the specified slide master.

`slideObj = add(presentation, slideLayout, index)` adds the slide at the index position specified by `index`, using the specified slide layout in the first slide master in the presentation.

`slideObj = add(presentation, slideLayout, slideMaster, index)` adds the slide at the index position specified by `index`, using the specified slide layout in the specified slide master.

## Examples

### Add Slides to a Presentation

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create and open a presentation named `myPresentation.pptx`.

```
ppt = Presentation('myPresentation.pptx');  
open(ppt);
```

Add the first slide and specify the slide layout, but not the slide master or location.

```
contentSlide = add(ppt, 'Title and Content');  
replace(contentSlide, 'Title', 'This is the Title of the Slide Content');
```

Add another slide using the Office Theme slide master. Insert it before the slide represented by contentSlide.

```
titleSlide = add(ppt, 'Title Slide', 'Office Theme', contentSlide);  
replace(titleSlide, 'Title', 'Presentation Title');
```

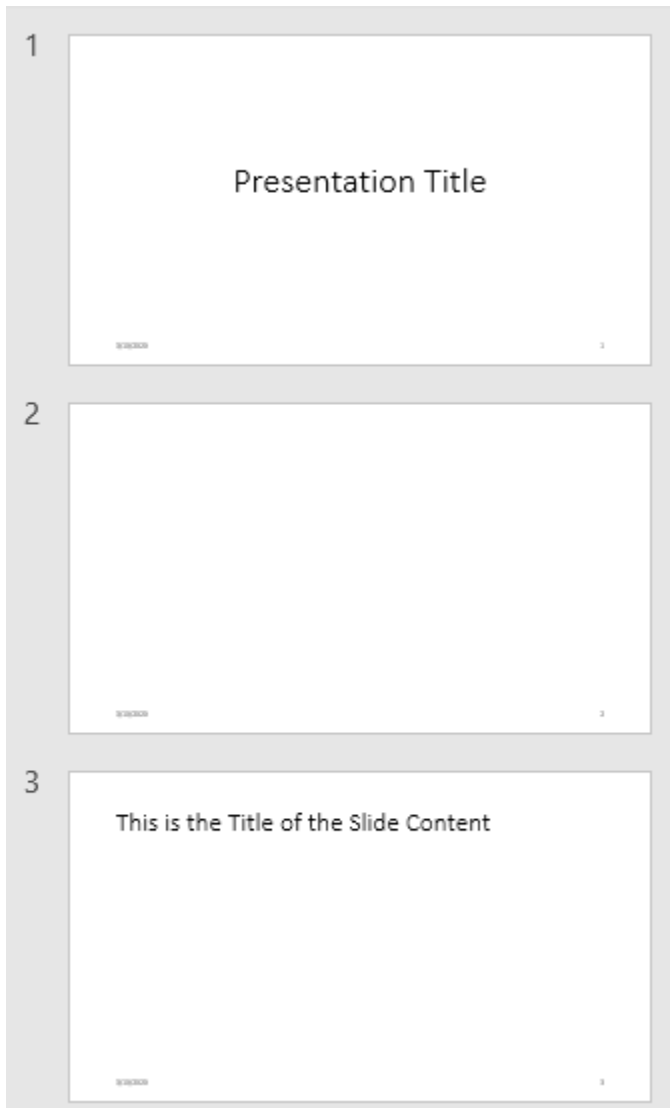
Add a blank slide using the Office Theme slide master. Make the new slide the second slide in the presentation.

```
blankSlide = add(ppt, 'Blank', 'Office Theme', 2);
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

Here is the generated presentation:



## Input Arguments

**presentation** — Presentation to add slide to

`mlreportgen.ppt.Presentation` object

Presentation to add content to, specified as an `mlreportgen.ppt.Presentation` object.

**slideLayout** — Layout of slide to add

character vector | string scalar

Layout of slide to add, specified as a character vector or string scalar. The layout must be in the presentation template.

To see the available layouts, you can:

- Use the `getLayoutNames` method.

- In the PowerPoint template, on the **Home** tab, in the **Slides** section, click **Layout**.

**slideMaster — Slide master for slide layout**

character vector | string scalar

Slide master for the specified slide layout, specified as a character vector or string scalar. The slide master must be in the presentation template.

To see the available slide masters, use one of these approaches:

- Use the `getMasterNames` method.
- In the PowerPoint template, on the **View** tab, in the **Master Views** section, click **Slide Master**. The slide masters are the numbered slides. Point to the slide master to see its name. Specify the name without including the words `Slide Master`.

**otherSlide — Slide to insert new slide before**

`mlreportgen.ppt.Slide` object

Slide to insert new slide before, specified as an `mlreportgen.ppt.Slide` object.

**index — Index position of slide in presentation**

integer

Index position of slide in presentation, specified as an integer.

## Output Arguments

**slideObj — Slide**

`mlreportgen.ppt.Slide` object

Slide, returned as an `mlreportgen.ppt.Slide` object.

## See Also

`replace` | `getLayoutNames` | `mlreportgen.ppt.Slide` | `mlreportgen.ppt.Presentation` | `getMasterNames`

## Topics

“Add and Replace Presentation Content” on page 14-58

## Introduced in R2015b

# close

**Class:** mlreportgen.ppt.Presentation

**Package:** mlreportgen.ppt

Close presentation

## Syntax

```
close(presentation)
```

## Description

`close(presentation)` closes the specified `mlreportgen.ppt.Presentation` object and generates the associated Microsoft PowerPoint presentation file.

## Examples

### Generate a Presentation

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation and add a title slide.

```
ppt = Presentation('myPresentation.pptx');  
open(ppt);  
slide = add(ppt, 'Title Slide');  
replace(slide, 'Title', 'My Title')
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

## Input Arguments

### **presentation** — Presentation to close and generate

`mlreportgen.ppt.Presentation` object

Presentation to close and generate, specified as an `mlreportgen.ppt.Presentation` object.

## See Also

`open` | `mlreportgen.ppt.Presentation`

### Topics

“Generate a Presentation” on page 14-13

**Introduced in R2015a**



# mlreportgen.ppt.Presentation.createTemplate

**Class:** mlreportgen.ppt.Presentation

**Package:** mlreportgen.ppt

Create copy of PPT API default presentation template

## Syntax

```
templatePath = mlreportgen.ppt.Presentation.createTemplate(path)
```

## Description

`templatePath = mlreportgen.ppt.Presentation.createTemplate(path)` creates a copy of the PPT API default presentation template at the specified location and returns the full path of the new template.

## Input Arguments

**path — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar. The .pptx file name extension is optional.

## Output Arguments

**templatePath — Path and file name of template copy**

character vector

Path and file name of the template copy, returned as a character vector

## Examples

### Create Copy of PPT API Default Template

Create a copy of the PPT API template in the current folder.

```
templatePath = mlreportgen.ppt.Presentation.createTemplate("MyTemplate.pptx");
```

## See Also

mlreportgen.ppt.Presentation

## Topics

“Set Up a PowerPoint Presentation Template” on page 14-23

**Introduced in R2021a**

## find

**Class:** mlreportgen.ppt.Presentation

**Package:** mlreportgen.ppt

Search a presentation

### Syntax

```
searchResults = find(presentation,objectName)
```

### Description

`searchResults = find(presentation,objectName)` searches the specified presentation for the content or slide objects whose Name property value matches `objectName`.

### Examples

#### Find Content in Presentation

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation. Add two slides that have titles.

```
ppt = Presentation('myPresentation.pptx');  
open(ppt);  
add(ppt,'Title Slide');  
add(ppt,'Title and Content');
```

Find presentation objects whose Name property is Title.

```
contents = find(ppt,'Title')
```

```
contents=1×2 object  
1×2 TextBoxPlaceholder array with properties:
```

```
Bold  
Font  
ComplexScriptFont  
FontColor  
FontSize  
Italic  
Strike  
Subscript  
Superscript  
Underline  
BackgroundColor  
VAlign  
Name
```

```
X  
Y  
Width  
Height  
Style  
Children  
Parent  
Tag  
Id
```

The `find` method returns a 1-by-2 array of `mlreportgen.ppt.TextBoxPlaceholder` objects. The first object is for the title in the first slide and the second object is for the title in the second slide.

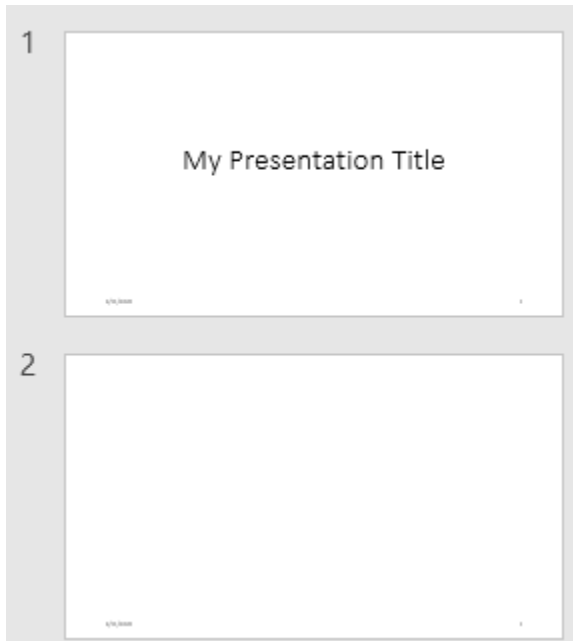
Replace the title in the first slide with `My Presentation Title`.

```
p = Paragraph('My Presentation Title');  
replace(contents(1),p);
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

Here is the generated presentation:



## Input Arguments

### **presentation** – Presentation to search

`mlreportgen.ppt.Presentation` object

Presentation to search, specified as an `mlreportgen.ppt.Presentation` object.

**objectName — Name property value to search for**

character vector | string scalar

Name property value to search for, specified as a character vector or string scalar.

When you add a slide to a presentation, the `add` method sets the `Name` property of the content objects on the slide based on the slide layout. See `getLayoutNames`. The `Name` property values include:

- 'Title'
- 'Content'
- 'Table'
- 'Picture'
- 'Text'
- 'Vertical Text'

The `Name` property can also be set to a custom value.

## Output Arguments

**searchResults — Search results**

array of content or slide objects

Search results, returned as an array of content or slide objects. The array can contain content objects, such as:

- `mlreportgen.ppt.ContentPlaceholder`
- `mlreportgen.ppt.TextBoxPlaceholder`
- `mlreportgen.ppt.TablePlaceholder`
- `mlreportgen.ppt.PicturePlaceholder`
- `mlreportgen.ppt.Table`
- `mlreportgen.ppt.Picture`

The array can also contain `mlreportgen.ppt.Slide` objects.

## See Also

`mlreportgen.ppt.Presentation` | `find`

### Topics

“Add and Replace Presentation Content” on page 14-58

### Introduced in R2015b

# getLayoutNames

**Class:** mlreportgen.ppt.Presentation

**Package:** mlreportgen.ppt

Get names of layouts for presentation slide master

## Syntax

```
layoutNames = getLayoutNames(presentation,slideMaster)
```

## Description

`layoutNames = getLayoutNames(presentation,slideMaster)` returns the names of layouts for a presentation slide master.

## Examples

### Get Names of Slide Layouts

Get the names of the slide layouts in the master slide of the default PPT API template. Use the layout names when you add slides to a presentation.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create and open a presentation.

```
ppt = Presentation('myPresentation.pptx');
open(ppt);
```

Get the names of the slide masters in the presentation. The default template has only one slide master.

```
masters = getMasterNames(ppt);
```

Get the names of layouts in the slide master.

```
layouts = getLayoutNames(ppt,masters{1})
```

```
layouts = 1×13 cell
    {'Title Slide'}    {'Title and Vertical Text'}    {'Vertical Title and Text'}    {'Title and
```

Add a title slide to the presentation and replace the title in the slide.

```
slide = add(ppt,'Title Slide');
replace(slide,'Title','My Title');
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

## Input Arguments

### **presentation** — Presentation to get layout names for

`mlreportgen.ppt.Presentation` object

Presentation to get layout names for, specified as an `mlreportgen.ppt.Presentation` object.

### **slideMaster** — Slide master to get layout names for

character vector | string scalar

Slide master to get layout names for, specified as a character vector or string scalar.

## Output Arguments

### **layoutNames** — Slide layout names

cell array of character vectors

Slide layout names, returned as a cell array of character vectors.

## See Also

`getMasterNames` | `getTableStyleNames` | `mlreportgen.ppt.Presentation`

## Topics

“Access PowerPoint Template Elements” on page 14-32

“Set Up a PowerPoint Presentation Template” on page 14-23

## Introduced in R2015b

# getMasterNames

**Class:** mlreportgen.ppt.Presentation

**Package:** mlreportgen.ppt

Get names of slide masters for presentation

## Syntax

```
masters = getMasterNames(presentation)
```

## Description

`masters = getMasterNames(presentation)` returns the names of slide masters for a presentation.

## Examples

### Get Slide Master Names in the Default Template

Get the slide master names so that you can find the layout names to use when you add slides to a presentation.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation.

```
ppt = Presentation('myPresentation');
open(ppt);
```

Get the names of the master slides in the presentation. The default PPT API template has only one slide master.

```
masters = getMasterNames(ppt);
masters{1}
```

```
ans =
'Office Theme'
```

Get the names of the layouts in the slide master.

```
layouts = getLayoutNames(ppt,masters{1})
```

```
layouts = 1x13 cell
    {'Title Slide'}    {'Title and Vertical Text'}    {'Vertical Title and Text'}    {'Title and
```

Add a title slide to the presentation, using the `Title Slide` layout, and replace the title in the slide.

```
slide = add(ppt, 'Title Slide');  
replace(slide, 'Title', 'My Title');
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

## Input Arguments

**presentation** — Presentation to get slide master names for

`mlreportgen.ppt.Presentation` object

Presentation to get slide master names for, specified as an `mlreportgen.ppt.Presentation` object.

## Output Arguments

**masters** — Slide masters in presentation

cell array of character vectors

Slide masters in the presentation, returned as a cell array of character vectors.

## See Also

`getLayoutNames` | `getTableStyleNames` | `mlreportgen.ppt.Presentation`

## Topics

“Access PowerPoint Template Elements” on page 14-32

“Set Up a PowerPoint Presentation Template” on page 14-23

**Introduced in R2015b**



# getTableStyleNames

**Class:** mlreportgen.ppt.Presentation

**Package:** mlreportgen.ppt

Get table style names for presentation

## Syntax

```
tableStyles = getTableStyleNames(presentation)
```

## Description

`tableStyles = getTableStyleNames(presentation)` gets the table style names for a presentation.

## Examples

### Get Table Style Names

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation.

```
ppt = Presentation('myPresentation.pptx');
open(ppt);
```

Get the names of the table styles in the presentation template.

```
getTableStyleNames(ppt)
```

```
ans = 74x2 cell
    {'Medium Style 2 - Accent 1'}    {'{5C22544A-7EE6-4342-B048-85BDC9FD1C3A}' }
    {'Light Style 1'}                {'{9D7B26C5-4107-4FEC-AEDC-1716B250A1EF}' }
    {'Light Style 1 - Accent 1'}    {'{3B4B98B0-60AC-42C2-AFA5-B58CD77FA1E5}' }
    {'Light Style 1 - Accent 2'}    {'{0E3FDE45-AF77-4B5C-9715-49D594BDF05E}' }
    {'Light Style 1 - Accent 3'}    {'{C083E6E3-FA7D-4D7B-A595-EF9225AFE82}' }
    {'Light Style 1 - Accent 4'}    {'{D27102A9-8310-4765-A935-A1911B00CA55}' }
    {'Light Style 1 - Accent 5'}    {'{5FD0F851-EC5A-4D38-B0AD-8093EC10F338}' }
    {'Light Style 1 - Accent 6'}    {'{68D230F3-CF80-4859-8CE7-A43EE81993B5}' }
    {'Light Style 2'}                {'{7E9639D4-E3E2-4D34-9284-5A2195B3D0D7}' }
    {'Light Style 2 - Accent 1'}    {'{69012ECD-51FC-41F1-AA8D-1B2483CD663E}' }
    {'Light Style 2 - Accent 2'}    {'{72833802-FEF1-4C79-8D5D-14CF1EAF98D9}' }
    {'Light Style 2 - Accent 3'}    {'{F2DE63D5-997A-4646-A377-4702673A728D}' }
    {'Light Style 2 - Accent 4'}    {'{17292A2E-F333-43FB-9621-5CBBE7FDCDCB}' }
    {'Light Style 2 - Accent 5'}    {'{5A111915-BE36-4E01-A7E5-04B1672EAD32}' }
    {'Light Style 2 - Accent 6'}    {'{912C8C85-51F0-491E-9774-3900AFEF0FD7}' }
    {'Light Style 3'}                {'{616DA210-FB5B-4158-B5E0-FEB733F419BA}' }
    {'Light Style 3 - Accent 1'}    {'{BC89EF96-8CEA-46FF-86C4-4CE0E7609802}' }
```

```

{'Light Style 3 - Accent 2' }    {'{5DA37D80-6434-44D0-A028-1B22A696006F}' }
{'Light Style 3 - Accent 3' }    {'{8799B23B-EC83-4686-B30A-512413B5E67A}' }
{'Light Style 3 - Accent 4' }    {'{ED083AE6-46FA-4A59-8FB0-9F97EB10719F}' }
{'Light Style 3 - Accent 5' }    {'{BDBED569-4797-4DF1-A0F4-6AAB3CD982D8}' }
{'Light Style 3 - Accent 6' }    {'{E8B1032C-EA38-4F05-BA0D-38AFFFC7BED3}' }
{'Medium Style 1'                }    {'{793D81CF-94F2-401A-BA57-92F5A7B2D0C5}' }
{'Medium Style 1 - Accent 1' }    {'{B301B821-A1FF-4177-AEE7-76D212191A09}' }
{'Medium Style 1 - Accent 2' }    {'{9DCAF9ED-07DC-4A11-8D7F-57B35C25682E}' }
{'Medium Style 1 - Accent 3' }    {'{1FECB4D8-DB02-4DC6-A0A2-4F2EBAE1DC90}' }
{'Medium Style 1 - Accent 4' }    {'{1E171933-4619-4E11-9A3F-F7608DF75F80}' }
{'Medium Style 1 - Accent 5' }    {'{FABFCF23-3B69-468F-B69F-88F6DE6A72F2}' }
{'Medium Style 1 - Accent 6' }    {'{10A1B5D5-9B99-4C35-A422-299274C87663}' }
{'Medium Style 2'                }    {'{073A0DAA-6AF3-43AB-8588-CEC1D06C72B9}' }
:

```

Create a table and specify the table style name.

```

table = Table({'a','b';'c','d'});
table.StyleName = 'Medium Style 2 - Accent 1';

```

Add a slide that has a title and a table. Replace the table placeholder with the table you created.

```

slide = add(ppt,'Title and Table');
replace(slide,'Table',table);

```

Close and view the presentation.

```

close(ppt);
rptview(ppt);

```

## Input Arguments

### presentation — Presentation to search for table style names

mlreportgen.ppt.Presentation object

Presentation to search for table style names, specified as an mlreportgen.ppt.Presentation object.

## Output Arguments

### tableStyles — Table styles in presentation

*N*-by-2 cell array of character vectors

Table styles in the presentation, returned as an *N*-by-2 cell array of character vectors. Each style has a name and a numeric identifier. For example:

```

{'Medium Style 2 - Accent 1'      }    {'{5C22544A-7EE6-4342-B048-85BDC9FD1C3A}' }

```

To specify a table style in an mlreportgen.ppt.Table object, you can use the name or numeric identifier. Use the numeric identifier when the name can vary based on locale. If you use the numeric identifier, include the curly braces inside of the single or double quotes. For example:

```

table = Table({'a','b';'c','d'});
table.StyleName = '{5C22544A-7EE6-4342-B048-85BDC9FD1C3A}';

```

## **See Also**

getMasterNames | getLayoutNames | mlreportgen.ppt.Presentation

## **Topics**

“Access PowerPoint Template Elements” on page 14-32

“Set Up a PowerPoint Presentation Template” on page 14-23

“Create and Format Tables” on page 14-69

## **Introduced in R2015b**

## open

**Class:** `mlreportgen.ppt.Presentation`

**Package:** `mlreportgen.ppt`

Open presentation

### Syntax

```
open(presentation)
```

### Description

`open(presentation)` opens a presentation and parses the template.

### Examples

#### Open a Presentation

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation.

```
ppt = Presentation('myPresentation.pptx');
```

Open the presentation.

```
open(ppt);
```

Add a title slide to the presentation.

```
slide = add(ppt, 'Title Slide');  
replace(slide, 'Title', 'My Title');
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

### Input Arguments

**presentation** — Presentation to open

`mlreportgen.ppt.Presentation` object

Presentation to open, specified as an `mlreportgen.ppt.Presentation` object.

## **See Also**

`close | mlreportgen.dom.Presentation`

## **Topics**

“Create a Presentation Generator” on page 14-2

“Generate a Presentation” on page 14-13

**Introduced in R2015a**

## replace

**Class:** `mlreportgen.ppt.Presentation`

**Package:** `mlreportgen.ppt`

Replace text, tables, or pictures in presentation

### Syntax

```
replace(presentation, contentName, content)
```

### Description

`replace(presentation, contentName, content)` replaces existing content in a presentation content object with the specified content which can be one or more paragraphs, a table, or a picture. If the type of content that you specify in the `content` argument is not valid for the content object identified by `contentName`, the `replace` method has no effect.

### Examples

#### Replace Presentation Content

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation and add two slides that have titles.

```
ppt = Presentation('myPresentation.pptx');  
open(ppt);  
add(ppt, 'Title Slide');  
add(ppt, 'Title and Content');
```

Replace all the titles in the presentation with the title `My Slide Title`.

```
replace(ppt, 'Title', 'My Slide Title');
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

### Input Arguments

**presentation** — Presentation to replace content in

`mlreportgen.ppt.Presentation` object

Presentation to replace content in, specified as an `mlreportgen.ppt.Presentation` object.

**contentName — Name property value of objects with content to replace**

character vector | string scalar

Name property value of the objects that contain the content that you want to replace, specified as a character vector or string scalar.

When you add a slide to a presentation, the `add` method sets the Name property of the content objects on the slide based on the slide layout. See `getLayoutNames`. The Name property values include:

- 'Title'
- 'Content'
- 'Table'
- 'Picture'
- 'Text'
- 'Vertical Text'

The Name property can also be set to a custom value.

**content — Replacement content**character vector | string scalar | `mlreportgen.ppt.Paragraph` object | cell array of character vectors or `Paragraph` objects | string array | `mlreportgen.ppt.Table` object | `mlreportgen.ppt.Picture` object

Replacement content, specified as one of these values:

- character vector or string scalar
- `mlreportgen.ppt.Paragraph` object
- cell array of character vectors or `Paragraph` objects, or a combination of both
- string array
- `mlreportgen.ppt.Table` object
- `mlreportgen.ppt.Picture` object

**Tips**

- To replace content on a specific slide, use the `replace` method of the `mlreportgen.ppt.Slide` object.

**See Also**`add` | `getLayoutNames` | `mlreportgen.ppt.Slide` | `mlreportgen.ppt.Presentation`**Topics**

"Add and Replace Presentation Content" on page 14-58

**Introduced in R2015b**

## formatAsHTML

**Class:** mlreportgen.ppt.ProgressMessage

**Package:** mlreportgen.ppt

Wrap message in HTML tags

### Syntax

```
htmlMessageOut = formatAsHTML(message)
```

### Description

`htmlMessageOut = formatAsHTML(message)` returns the message text formatted with HTML tags.

### Examples

#### Format a Message as HTML

This example uses `formatAsHTML` to display the progress messages.

```
import mlreportgen.ppt.*;
pre = Presentation('myPresentation.pptx');

dispatcher = MessageDispatcher.getTheDispatcher;
l = addlistener(dispatcher, 'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsHTML));

dispatch(dispatcher, ErrorMessage('invalid slide', pre));
open(pre);

titleText = Text('This is a Title');
titleText.Style = {Bold};

replace(pre, 'Title', titleText);

close(pre);

delete(l);
```

### Input Arguments

**message** — Progress message

mlreportgen.ppt.ProgressMessage object

Progress message, specified as an `mlreportgen.ppt.ProgressMessage` object.



## Output Arguments

### **htmlMessageOut — Progress message with HTML tagging**

`mlreportgen.ppt.ProgressMessage` object

Progress message with HTML tagging, returned as an `mlreportgen.ppt.ProgressMessage` object.

### **See Also**

`formatAsText` | `mlreportgen.ppt.ProgressMessage` | `mlreportgen.ppt.MessageFilter` | `mlreportgen.ppt.MessageEventData`

### **Topics**

“Display Presentation Generation Messages” on page 14-14

### **Introduced in R2015b**

## formatAsText

**Class:** mlreportgen.ppt.ProgressMessage

**Package:** mlreportgen.ppt

Format message as text

### Syntax

```
textMessageOut = formatAsText(message)
```

### Description

`textMessageOut = formatAsText(message)` returns the message text formatted as text.

### Examples

#### Format a Message Text

This example uses `formatAsText` to display the progress messages.

```
import mlreportgen.ppt.*;
pre = Presentation('myPresentation.pptx');

dispatcher = MessageDispatcher.getTheDispatcher;
dispatcher.Filter.DebugMessagesPass = true;
l = addlistener(dispatcher, 'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsText));

dispatch(dispatcher, ErrorMessage('invalid slide', pre));
open(pre);

titleText = Text('This is a Title');
titleText.Style = {Bold};

replace(pre, 'Title', titleText);

close(pre);

delete(l);
```

### Input Arguments

#### message — The PPT progress message

mlreportgen.ppt.ProgressMessage object

The PPT progress message, specified as an `mlreportgen.ppt.ProgressMessage` object.

## Output Arguments

### **textMessageOut — PPT progress message formatted as text**

`mlreportgen.ppt.ProgressMessage` object

PPT progress message formatted as text, returned as an `mlreportgen.ppt.ProgressMessage` object.

## See Also

`formatAsHTML` | `mlreportgen.ppt.ProgressMessage` | `mlreportgen.ppt.MessageFilter` | `mlreportgen.ppt.MessageEventData`

## Topics

“Display Presentation Generation Messages” on page 14-14

## Introduced in R2015b

## passesFilter

**Class:** mlreportgen.ppt.ProgressMessage

**Package:** mlreportgen.ppt

Determine if message passes filter

### Syntax

```
tf = passesFilter(message,filter)
```

### Description

`tf = passesFilter(message,filter)` determines whether the message passes the filter.

### Examples

#### Determine Whether a Message Passes a Filter

This example shows how to add a progress message to display when generating a presentation.

Add a dispatcher and listener to the report. Configure the dispatcher to include debug messages.

```
import mlreportgen.ppt.*;
pre = Presentation('myPresentation.pptx');

dispatcher = MessageDispatcher.getTheDispatcher;
dispatcher.Filter.DebugMessagesPass = true;
l = addlistener(dispatcher,'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsText));
```

Create a progress message.

```
dispatch(dispatcher,ErrorMessage('invalid slide',pre));
open(pre);
```

```
titleText = Text('This is a Title');
titleText.Style = {Bold};
```

```
replace(pre,'Title',titleText);
```

Generate the presentation and delete the listener.

```
close(pre);
```

```
delete(l);
```

Check the progress messages in the MATLAB Command Window. In addition to the predefined PPT progress messages, the `starting` chapter message added in this example appears. The output also includes debug messages.

## Input Arguments

### **message** — PPT progress message

`mlreportgen.ppt.ProgressMessage` object

PPT progress message, specified as an `mlreportgen.ppt.ProgressMessage` object.

### **filter** — Filter to use with message

`mlreportgen.ppt.MessageFilter` object

Filter to use with the progress message, specified as an `mlreportgen.ppt.MessageFilter` object.

## Output Arguments

### **tf** — Indication of whether the message passes the filter

Boolean

- 1 — Messages passes the specified filter (the dispatcher handles the message)
- 0 — Messages fails the specified filter (the dispatcher ignores the message)

## See Also

`mlreportgen.ppt.MessageFilter` | `mlreportgen.ppt.ProgressMessage` | `mlreportgen.ppt.MessageEventData`

## Topics

“Display Presentation Generation Messages” on page 14-14

**Introduced in R2015b**

## add

**Class:** `mlreportgen.ppt.Slide`

**Package:** `mlreportgen.ppt`

Add text box, table, or picture to slide

### Syntax

```
addedObj = add(slide,object)
```

### Description

`addedObj = add(slide,object)` adds the specified `mlreportgen.ppt.TextBox`, `mlreportgen.ppt.Table`, or `mlreportgen.ppt.Picture` object to a slide.

### Examples

#### Add a Picture to a Slide

Use the `add` method of an `mlreportgen.ppt.Presentation` object to add a slide to a presentation. Then, use the `add` method of the resulting `mlreportgen.ppt.Slide` object to add a picture to the slide.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation and add a slide to it. This example adds a slide with a `Blank` layout.

```
ppt = Presentation('mySlideAddPresentation.pptx');  
slide = add(ppt, 'Blank');
```

Create an `mlreportgen.ppt.Picture` object for the picture that you want to add to the slide. Use the `Picture` object properties to specify the size of the picture in the slide.

```
plane = Picture('b747.jpg');  
plane.X = '4in';  
plane.Y = '4in';  
plane.Width = '5in';  
plane.Height = '2in';
```

Add the `Picture` object to the slide.

```
add(slide,plane);
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

Here is the generated presentation:



## Input Arguments

### **slide** — Slide to add content to

`mreportgen.ppt.Slide` object

Slide to add content to, specified as an `mreportgen.ppt.Slide` object.

### **object** — Object to add to slide

`mreportgen.ppt.Picture` | `mreportgen.ppt.Table` | `mreportgen.ppt.TextBox`

Object to add to a slide, specified as one of these PPT API objects:

- `mreportgen.ppt.Picture`
- `mreportgen.ppt.Table`
- `mreportgen.ppt.TextBox`

## Output Arguments

### **addedObj** — Object added to slide

`mreportgen.ppt.Picture` | `mreportgen.ppt.Table` | `mreportgen.ppt.TextBox`

Object added to slide, returned as one of these PPT API objects:

- `mreportgen.ppt.Picture`
- `mreportgen.ppt.Table`
- `mreportgen.ppt.TextBox`

## See Also

`mreportgen.ppt.Slide` | `mreportgen.ppt.Table` | `mreportgen.ppt.TextBox` | `mreportgen.ppt.Picture`

## Topics

“Add and Replace Presentation Content” on page 14-58

**Introduced in R2015b**

## find

**Class:** mlreportgen.ppt.Slide

**Package:** mlreportgen.ppt

Search slide for content

### Syntax

```
searchResults = find(slide,objectName)
```

### Description

`searchResults = find(slide,objectName)` searches a slide for a slide content object whose Name property value matches `objectName`.

### Examples

#### Search a Slide for Content

Find a content object in an `mlreportgen.ppt.Slide` object by using the `find` method.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation.

```
ppt = Presentation('mySlideFindPresentation.pptx');  
open(ppt);
```

Add a slide with a Title and Content layout.

```
slide = add(ppt,'Title and Content');
```

Search the slide for a content object that has the Name property value 'Content'.

```
contents = find(slide,'Content')
```

```
contents =  
    ContentPlaceholder with properties:
```

```
        Bold: []  
        Font: []  
ComplexScriptFont: []  
        FontColor: []  
        FontSize: []  
        Italic: []  
        Strike: []  
        Subscript: []  
        Superscript: []
```



```

    Underline: []
    BackgroundColor: []
    VAlign: []
    Name: 'Content'
    X: []
    Y: []
    Width: []
    Height: []
    Style: []
    Children: []
    Parent: [1x1 mlreportgen.ppt.Slide]
    Tag: 'ppt.ContentPlaceholder:375:99'
    Id: '375:99'

```

`find` returns a 1-by-1 array that contains an `mlreportgen.ppt.ContentPlaceholder` object. Specify that the text in the placeholder object is bold and add text to the object.

```

contents(1).Bold = true;
add(contents(1), 'This is bold text');

```

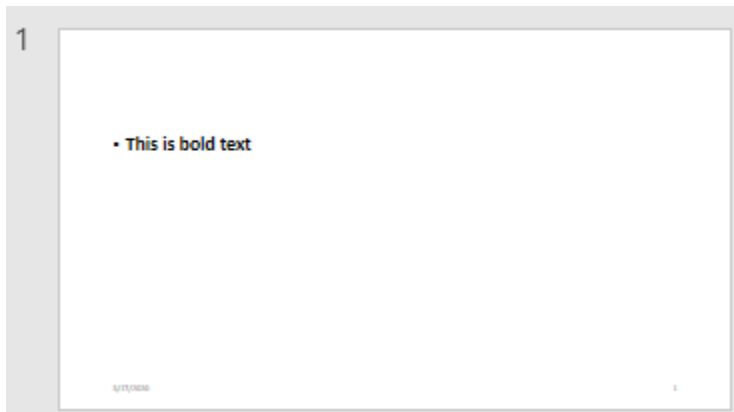
Close and view the presentation.

```

close(ppt);
rptview(ppt);

```

Here is the generated presentation:



## Input Arguments

### **slide** — Slide to search

`mlreportgen.ppt.Slide` object

Slide to search, specified as an `mlreportgen.ppt.Slide` object.

### **objectName** — Name property value to search for

character vector | string scalar

Name property value to search for, specified as a character vector or string scalar.

When you add a slide to a presentation, the `add` method sets the `Name` property of the content objects in the slide based on the slide layout. See `getLayoutNames`. The `Name` property values include:

- `'Title'`
- `'Content'`
- `'Table'`
- `'Picture'`
- `'Text'`
- `'Vertical Text'`

The `Name` property can also be set to a custom value.

## Output Arguments

### **searchResults** — Search results

array of content objects

Search results, returned as an array of content objects such as:

- `mlreportgen.ppt.ContentPlaceholder`
- `mlreportgen.ppt.TextBoxPlaceholder`
- `mlreportgen.ppt.TablePlaceholder`
- `mlreportgen.ppt.PicturePlaceholder`
- `mlreportgen.ppt.Table`
- `mlreportgen.ppt.Picture`

## See Also

`mlreportgen.ppt.Slide` | `find`

## Topics

“Add and Replace Presentation Content” on page 14-58

## Introduced in R2015b

# replace

**Class:** mlreportgen.ppt.Slide

**Package:** mlreportgen.ppt

Replace text, tables, or pictures in a slide

## Syntax

```
replace(slide, contentName, content)
```

## Description

`replace(slide, contentName, content)` replaces the existing content in a slide content object with the specified content, which can be one or more paragraphs, a table, or a picture. If the type of content that you specify in the `content` argument is not valid for the content object identified by `contentName`, the `replace` method has no effect.

## Examples

### Replace Slide Content

Use the `replace` method of an `mlreportgen.ppt.Slide` object to replace placeholder content with your content.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation. Add a slide for text, a slide for a picture, and a slide for a table.

```
ppt = Presentation('mySlideReplacePresentation.pptx');  
slide1 = add(ppt, 'Title and Content');  
slide2 = add(ppt, 'Title and Picture');  
slide3 = add(ppt, 'Title and Table');
```

In the first slide, replace the `Title` and `Content` placeholders with text.

```
replace(slide1, 'Title', 'Text Slide');  
replace(slide1, 'Content', 'This is the content for slide 1');
```

In the second slide, replace the `Title` placeholder with text and the `Picture` placeholder with a picture.

```
replace(slide2, 'Title', 'Picture Slide');  
replace(slide2, 'Picture', Picture('b747.jpg'));
```

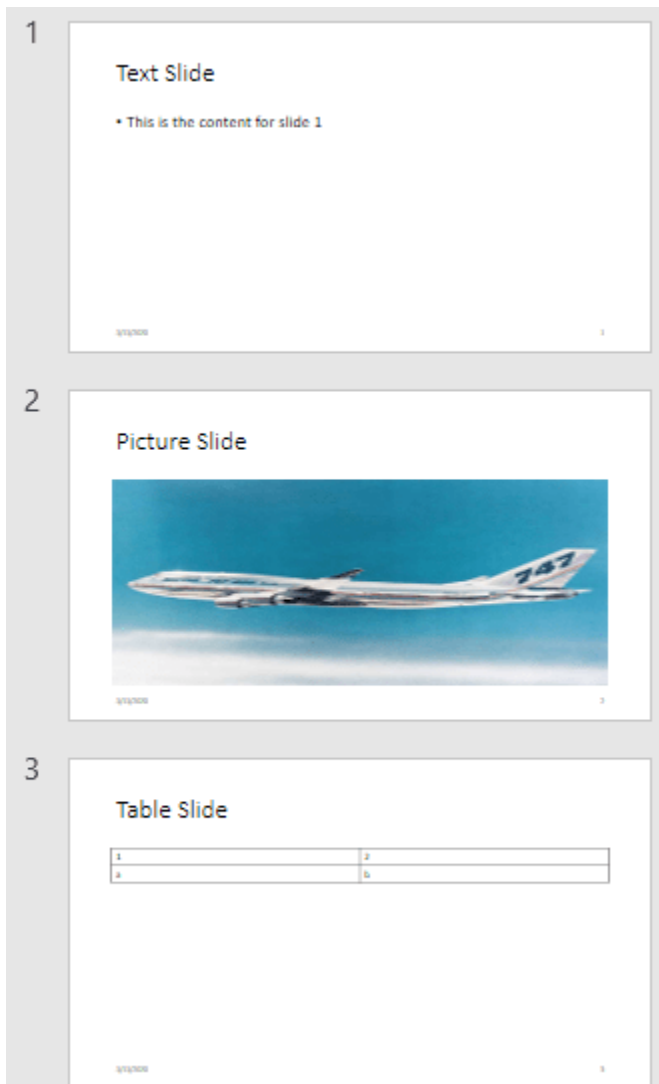
In the third slide, replace the `Title` placeholder with text and the `Table` placeholder with a table.

```
replace(slide3, 'Title', 'Table Slide');  
replace(slide3, 'Table', Table({1 2; 'a' 'b'}));
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

Here is the generated presentation:



## Input Arguments

**slide** — Slide to replace content in  
mlreportgen.ppt.Slide object

Slide to replace content in, specified as an mlreportgen.ppt.Slide object.

**contentName** — Name property value of object with content to replace  
character vector | string scalar

Name property value of the object that contains the content that you want to replace, specified as a character vector or string scalar.

When you add a slide to a presentation, the `add` method sets the Name property of the content objects on the slide based on the slide layout. See `getLayoutNames`. The Name property values include:

- 'Title'
- 'Content'
- 'Table'
- 'Picture'
- 'Text'
- 'Vertical Text'

The Name property can also be set to a custom value.

### **content — Replacement content**

character vector | string scalar | `mlreportgen.ppt.Paragraph` object | string array | `mlreportgen.ppt.Table` object | `mlreportgen.ppt.Picture` object | cell array

Replacement content, specified as one of these values:

- character vector or string scalar
- `mlreportgen.ppt.Paragraph` object
- string array
- `mlreportgen.ppt.Table` object
- `mlreportgen.ppt.Picture` object
- cell array of character vectors or `Paragraph` objects, or a combination of character vectors, strings scalars, string arrays, or `mlreportgen.ppt.Paragraph` objects. Contents of inner cell arrays or string arrays are indented from the contents of the outer cell array.

### **See Also**

`mlreportgen.ppt.Slide`

### **Topics**

“Add and Replace Presentation Content” on page 14-58

“Create a Presentation Programmatically” on page 14-49

### **Introduced in R2015b**

## append

**Class:** mlreportgen.ppt.Table

**Package:** mlreportgen.ppt

Append row to table

### Syntax

```
rowObj = append(table, row)
```

### Description

`rowObj = append(table, row)` appends a row to a table.

### Examples

#### Create a Table with Table Rows

Create a presentation.

```
import mlreportgen.ppt.*
```

```
ppt = Presentation('myTableEntryPresentation.pptx');  
open(ppt);  
add(ppt, 'Title and Content');
```

Create a table with three columns.

```
table1 = Table();
```

Create the first table row.

```
tr1 = TableRow();  
tr1.Style = {Bold(true)};
```

Create three table entries for the first row.

```
te1tr1 = TableEntry();  
p = Paragraph('first entry');  
p.FontColor = 'red';  
append(te1tr1, p);
```

```
te2tr1 = TableEntry();  
append(te2tr1, 'second entry');
```

```
te3tr1 = TableEntry();  
te3tr1.Style = {FontColor('green')};  
append(te3tr1, 'third entry');
```

Append the table entries to the first row.

```
append(tr1,te1tr1);
append(tr1,te2tr1);
append(tr1,te3tr1);
```

Create the second table row.

```
tr2 = TableRow();
```

Create three table entries for the second row.

```
te1tr2 = TableEntry();
te1tr2.Style = {FontColor('red')};
p = Paragraph('first entry');
append(te1tr2,p);
```

```
te2tr2 = TableEntry();
append(te2tr2,'second entry');
```

```
te3tr2 = TableEntry();
te3tr2.Style = {FontColor('green')};
append(te3tr2,'third entry');
```

Append the table entries to the second row.

```
append(tr2,te1tr2);
append(tr2,te2tr2);
append(tr2,te3tr2);
```

Append the table rows to the table.

```
append(table1,tr1);
append(table1,tr2);
```

Use the `mlreportgen.ppt.Presentation.find` method to find the slides that have a Content placeholder. In this case, there are two.

```
contents = find(ppt,'Content');
```

Replace the table in the second slide with `table1`.

```
replace(contents(1),table1);
```

Generate the presentation. Open the `myTableEntryPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
close(ppt);
rptview(ppt);
```

1

first entry	second entry	third entry
first entry	second entry	third entry

6/11/2015 3

## Input Arguments

### **table** — Table to append row to

`mlreportgen.ppt.Table` object

Table to append row to, specified as an `mlreportgen.ppt.Table` object.

### **row** — Row to append to table

`mlreportgen.ppt.TableRow` object

Row to append to table, specified as an `mlreportgen.ppt.TableRow` object.

## Output Arguments

### **rowObj** — Appended table row

`mlreportgen.dom.TableRow` object

Appended table row, returned as an `mlreportgen.dom.TableRow` object.

## See Also

`mlreportgen.ppt.Table` | `mlreportgen.ppt.TableRow` | `mlreportgen.ppt.TableEntry`

## Topics

“Create and Format Tables” on page 14-69

## Introduced in R2015b



# entry

**Class:** mlreportgen.ppt.Table

**Package:** mlreportgen.ppt

Access table entry

## Syntax

```
tableEntryOut = entry(tableObj,row,column)
```

## Description

`tableEntryOut = entry(tableObj, row, column)` returns the table entry for the specified column of the specified row.

## Examples

### Color a Table Entry

Color the table entry in row 3, column 4.

```
import mlreportgen.ppt.*;

ppt = Presentation('myTableEntryMethod.pptx');
open(ppt);
slide1 = add(ppt, 'Title and Content');
t = Table(magic(5));
entry4row3 = t.entry(3,4);
entry4row3.BackgroundColor = 'red';

replace(slide1, 'Content', t);

close(ppt);
rptview(ppt);
```

1

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

7/6/2015 1

## Input Arguments

### **tableObj** — Table containing entry

`mlreportgen.ppt.Table` object

Table containing the entry, specified as an `mlreportgen.ppt.Table` object.

### **row** — Table row containing entry

double

Table row containing the entry, specified as a double. The double is an index number indicating the position of the row. The number of the top row is 1.

Data Types: double

### **column** — Column containing entry

double

Table column containing the entry, specified as a double. The double is an index number indicating the position of the column. The number of the left column is 1.

Data Types: double

## Output Arguments

### **tableEntryOut** — Table entry object

`mlreportgen.ppt.TableEntry` object

Table entry object, returned as an `mlreportgen.ppt.TableEntry` object

## See Also

`mlreportgen.ppt.TableEntry` | `row`

**Topics**

“Create and Format Tables” on page 14-69

**Introduced in R2014b**

## replace

**Class:** mlreportgen.ppt.Table

**Package:** mlreportgen.ppt

Replace table with another table

### Syntax

```
tableObj = replace(table, replacementTable)
```

### Description

tableObj = replace(table, replacementTable) replaces a table with another table.

### Examples

#### Replace Table with Another Table

Create a presentation.

```
import mlreportgen.ppt.*  
  
ppt = Presentation("myTableReplacePresentation.pptx");  
open(ppt);  
slide1 = add(ppt, 'Blank');
```

Create an mlreportgen.ppt.Table object.

```
t1 = Table(magic(7));  
t1.X = '2in';  
t1.Y = '2in';  
t1.Width = '6in';  
t1.Height = '4in';
```

Search in slide1 for Table.

```
add(slide1, t1);
```

Create another mlreportgen.ppt.Table object.

```
t2 = Table(magic(9));  
t2.X = '2in';  
t2.Y = '2in';  
t2.Width = '7in';  
t2.Height = '5in';
```

Replace t1 with t2.

```
replace(t1, t2);
```

Generate the presentation. Open myTableReplacePresentation.pptx. On a Windows platform, you can open the presentation in MATLAB:

```
close(ppt);  
rptview(ppt);
```

## Input Arguments

### **table** — Table to replace with another table

`mlreportgen.ppt.Table` object

Table to replace table, specified as an `mlreportgen.ppt.Table` object.

### **replacementTable** — Table to use as replacement

`mlreportgen.ppt.Table` object

Table to use as replacement, specified as an `mlreportgen.ppt.Table` object.

## Output Arguments

### **tableObj** — Table

`mlreportgen.ppt.Table` object

Table, returned as an `mlreportgen.ppt.Table` object.

## See Also

`mlreportgen.ppt.Table` | `mlreportgen.ppt.TablePlaceholder`

### Topics

“Add or Replace a Table” on page 14-61

“Create and Format Tables” on page 14-69

### Introduced in R2015b

## row

**Class:** mlreportgen.ppt.Table

**Package:** mlreportgen.ppt

Access table row

### Syntax

```
tableRowOut = row(table,rowNumber)
```

### Description

`tableRowOut = row(table,rowNumber)` returns the row specified by the `rowNumber`.

### Examples

#### Color a Table Row

Color the third row in the table.

```
import mlreportgen.ppt.*;

ppt = Presentation('myTableRowMethod.pptx');
open(ppt);
slide1 = add(ppt,'Title and Content');
t = Table(magic(5));
row3 = row(t,3);
row3.BackgroundColor = 'red';

replace(slide1,'Content',t);

close(ppt);
rptview(ppt);
```

### Input Arguments

**table** — Table containing row

mlreportgen.ppt.Table object

Table containing the row, specified as an mlreportgen.ppt.Table object.

**rowNumber** — Table row

double

Table row, specified as a double. The double is an index number indicating the position of the row. The number of the top row is 1.

Data Types: double

---

## Output Arguments

**tableRowOut — Table row object**

`mlreportgen.ppt.TableRow` object

Table row object, returned as an `mlreportgen.ppt.TableRow` object

## See Also

`mlreportgen.ppt.TableRow` | entry

## Topics

“Create and Format Tables” on page 14-69

## Introduced in R2014b

## append

**Class:** mlreportgen.ppt.TableEntry

**Package:** mlreportgen.ppt

Append text or paragraph to table entry

### Syntax

```
tableEntryObj = append(tableEntry, content)
```

### Description

`tableEntryObj = append(tableEntry, content)` appends text or a Paragraph object to a table entry.

### Examples

#### Create a Table with Table Rows and Entries

Create a presentation.

```
import mlreportgen.ppt.*  
  
ppt = Presentation('myTableEntryPresentation.pptx');  
open(ppt);  
add(ppt, 'Title and Content');
```

Create a table with three columns.

```
table1 = Table(3);
```

Create the first table row.

```
tr1 = TableRow();  
tr1.Style = {Bold(true)};
```

Create three table entries for the first row.

```
te1tr1 = TableEntry();  
p = Paragraph('first entry');  
p.FontColor = 'red';  
append(te1tr1, p);  
  
te2tr1 = TableEntry();  
append(te2tr1, 'second entry');  
  
te3tr1 = TableEntry();  
te3tr1.Style = {FontColor('green')};  
append(te3tr1, 'third entry');
```

Append the table entries to the first row.



```
append(tr1,te1tr1);
append(tr1,te2tr1);
append(tr1,te3tr1);
```

Create the second table row.

```
tr2 = TableRow();
```

Create three table entries for the second row.

```
te1tr2 = TableEntry();
te1tr2.Style = {FontColor('red')};
p = Paragraph('first entry');
append(te1tr2,p);
```

```
te2tr2 = TableEntry();
append(te2tr2,'second entry');
```

```
te3tr2 = TableEntry();
te3tr2.Style = {FontColor('green')};
append(te3tr2,'third entry');
```

Append the table entries to the second row.

```
append(tr2,te1tr2);
append(tr2,te2tr2);
append(tr2,te3tr2);
```

Append the table rows to the table.

```
append(table1,tr1);
append(table1,tr2);
```

Use the `mlreportgen.ppt.Presentation.find` method to find the slides that have a Content placeholder. In this case, there are two.

```
contents = find(ppt,'Content');
```

Replace the table in the second slide with `table1`.

```
replace(contents(1),table1);
```

Generate the presentation. Open `myTableEntryPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
close(ppt);
rptview(ppt);
```

1

first entry	second entry	third entry
first entry	second entry	third entry

6/11/2015 3

## Input Arguments

**tableEntry** — Table entry to append content to  
`mlreportgen.ppt.TableEntry` object

Table entry to append content to, specified as an `mlreportgen.ppt.TableEntry` object.

**content** — Content to append to table entry  
 character vector | `mlreportgen.ppt.Paragraph` object

Content to append to a table entry, specified as a character vector or one or more `mlreportgen.ppt.Paragraph` objects.

## Output Arguments

**paragraph** — Content appended to table entry  
`mlreportgen.ppt.Paragraph` object

Content appended to table entry, returned as an `mlreportgen.ppt.Paragraph` object.

## See Also

`mlreportgen.ppt.Table` | `mlreportgen.ppt.TableRow` | `mlreportgen.ppt.TableEntry` | `mlreportgen.ppt.Paragraph`

## Topics

“Add or Replace a Table” on page 14-61

“Create and Format Tables” on page 14-69

**Introduced in R2015b**

# replace

**Class:** mlreportgen.ppt.TablePlaceholder

**Package:** mlreportgen.ppt

Replace table placeholder with table

## Syntax

```
replacementTableObj = replace(tablePlaceholder, table)
```

## Description

`replacementTableObj = replace(tablePlaceholder, table)` replaces the specified table placeholder with the specified table and returns the table object.

## Examples

### Replace Table Placeholder with Table

Add a `Title` and `Table` slide to a presentation and then replace the title and table placeholders with your own title and table.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation.

```
ppt = Presentation("myTablePlaceholderPresentation.pptx");  
open(ppt);
```

Add a slide that has a `Title` and `Table` layout.

```
slide = add(ppt, "Title and Table");
```

Use the `find` method of the slide object to find the placeholder object that has the name `Title`.

```
titlePlaceholderObj = find(slide, "Title")
```

```
titlePlaceholderObj =  
    TextBoxPlaceholder with properties:
```

```
        Bold: []  
        Font: []  
ComplexScriptFont: []  
        FontColor: []  
        FontSize: []  
        Italic: []  
        Strike: []  
        Subscript: []
```

```
Superscript: []
Underline: []
BackgroundColor: []
VAlign: []
Name: 'Title'
X: []
Y: []
Width: []
Height: []
Style: []
Children: []
Parent: [1x1 mlreportgen.ppt.Slide]
Tag: 'ppt.TextBoxPlaceholder:30:96'
Id: '30:96'
```

The `find` method returns an `mlreportgen.ppt.TextBoxPlaceholder` object.

Replace the placeholder content with the title text.

```
replace(titlePlaceholderObj, "Fourth-Order Magic Square");
```

Use the `find` method of the slide object to find the placeholder object that has the name `Table`.

```
tablePlaceholderObj = find(slide, "Table")
```

```
tablePlaceholderObj =
```

```
TablePlaceholder with properties:
```

```
    Bold: []
    Font: []
ComplexScriptFont: []
    FontColor: []
    FontSize: []
    Italic: []
    Strike: []
    Subscript: []
    Superscript: []
    Underline: []
BackgroundColor: []
VAlign: []
Name: 'Table'
X: []
Y: []
Width: []
Height: []
Style: []
Children: []
Parent: [1x1 mlreportgen.ppt.Slide]
Tag: 'ppt.TablePlaceholder:31:97'
Id: '31:97'
```

The `find` method returns an `mlreportgen.ppt.TablePlaceholder` object.

Replace the table placeholder with a table for a fourth-order magic square.

```
replace(tablePlaceholderObj, Table(magic(4)));
```

Close and view the presentation

```
close(ppt);
rptview(ppt);
```

Here is the slide in the generated presentation:

16	2	3	13
5	11	10	8
9	7	6	12
4	14	15	1

## Input Arguments

### **tablePlaceholder** — Table placeholder

`mlreportgen.ppt.TablePlaceholder` object

Table placeholder, specified as an `mlreportgen.ppt.TablePlaceholder` object.

### **table** — Table to use as replacement

`mlreportgen.ppt.Table` object

Table to use as the replacement for the table placeholder, specified as an `mlreportgen.ppt.Table` object.

## Output Arguments

### **replacementTableObj** — Replacement table

`mlreportgen.ppt.Table` object

Replacement table, returned as an `mlreportgen.ppt.Table` object.

## See Also

`mlreportgen.ppt.TablePlaceholder` | `mlreportgen.ppt.Table` | `mlreportgen.ppt.TemplateTable`

## Topics

“Access PowerPoint Template Elements” on page 14-32

“Add or Replace a Table” on page 14-61

**Introduced in R2015b**

# append

**Class:** mlreportgen.ppt.TableRow

**Package:** mlreportgen.ppt

Append table entry to table row

## Syntax

```
tableEntryObj = append(tableRow,entry)
```

## Description

tableEntryObj = append(tableRow,entry) appends a table entry to a table row.

## Examples

### Create a Table with Table Rows and Entries

Create a presentation.

```
import mlreportgen.ppt.*  
  
ppt = Presentation('myTableEntryPresentation.pptx');  
open(ppt);  
add(ppt,'Title and Content');
```

Create a table with three columns.

```
table1 = Table(3);
```

Create the first table row.

```
tr1 = TableRow();  
tr1.Style = {Bold(true)};
```

Create three table entries for the first row.

```
te1tr1 = TableEntry();  
p = Paragraph('first entry');  
p.FontColor = 'red';  
append(te1tr1,p);  
  
te2tr1 = TableEntry();  
append(te2tr1,'second entry');  
  
te3tr1 = TableEntry();  
te3tr1.Style = {FontColor('green')};  
append(te3tr1,'third entry');
```

Append the table entries to the first row.

```
append(tr1,te1tr1);
append(tr1,te2tr1);
append(tr1,te3tr1);
```

Create the second table row.

```
tr2 = TableRow();
```

Create three table entries for the second row.

```
te1tr2 = TableEntry();
te1tr2.Style = {FontColor('red')};
p = Paragraph('first entry');
append(te1tr2,p);
```

```
te2tr2 = TableEntry();
append(te2tr2,'second entry');
```

```
te3tr2 = TableEntry();
te3tr2.Style = {FontColor('green')};
append(te3tr2,'third entry');
```

Append the table entries to the second row.

```
append(tr2,te1tr2);
append(tr2,te2tr2);
append(tr2,te3tr2);
```

Append the table rows to the table.

```
append(table1,tr1);
append(table1,tr2);
```

Use the `mlreportgen.ppt.Presentation.find` method to find the slides that have a Content placeholder. In this case, there are two.

```
contents = find(ppt,'Content');
```

Replace the table in the second slide with `table1`.

```
replace(contents(1),table1);
```

Generate the presentation. Open `myTableEntryPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
close(ppt);
rptview(ppt);
```



1

first entry	second entry	third entry
first entry	second entry	third entry

6/11/2015 3

## Input Arguments

**tableRow** — Table row to append content to  
`mlreportgen.ppt.TableRow` object

Table row to append content to, specified as an `mlreportgen.ppt.TableRow` object.

**entry** — Table entry to append to table row  
`mlreportgen.ppt.TableEntry` object

Table entry to append to a table row, specified as an `mlreportgen.ppt.TableEntry` object.

## Output Arguments

**tableEntryObj** — Table entry  
`mlreportgen.dom.TableEntry` object

Table entry, returned as an `mlreportgen.dom.TableEntry` object.

## See Also

`mlreportgen.ppt.Table` | `mlreportgen.ppt.TableRow` | `mlreportgen.ppt.TableEntry`

## Topics

“Add or Replace a Table” on page 14-61

“Create and Format Tables” on page 14-69

**Introduced in R2015b**

## replace

**Class:** `mlreportgen.ppt.TemplatePicture`

**Package:** `mlreportgen.ppt`

Replace template picture with another picture

### Syntax

```
newPicture = replace(templatePicture, replacementPicture)
```

### Description

`newPicture = replace(templatePicture, replacementPicture)` replaces a template picture with another picture.

### Input Arguments

**templatePicture — Template picture to replace**

`mlreportgen.ppt.TemplatePicture` object

Template picture to replace, specified as an `mlreportgen.ppt.TemplatePicture` object.

**replacementPicture — Replacement picture**

`mlreportgen.ppt.Picture` object

Replacement picture, specified as an `mlreportgen.ppt.Picture` object.

### Output Arguments

**newPicture — New picture**

`mlreportgen.ppt.Picture` object

New picture in the presentation slide, specified as an `mlreportgen.ppt.Picture` object.

## Examples

### Replace Template Picture

Generate a presentation, `MyPicturePresentation`, that you then use as the template presentation for another presentation. `MyPicturePresentation` has one slide with one picture.

```
import mlreportgen.ppt.*
ppt = Presentation("MyPicturePresentation");
open(ppt);
slidel = add(ppt, "Title and Picture");
replace(slidel, "Title", "Street");
replace(slidel, "Picture", Picture("street1.jpg"));
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Create a presentation, `MyNewPicturePresentation`, from `MyPicturePresentation`. `MyPicturePresentation` is the template presentation for `MyNewPicturePresentation`,

```
ppt = Presentation("MyNewPicturePresentation", "MyPicturePresentation");
open(ppt);
```

Find the template picture by using the `find` method of the slide object. Because the picture comes from a template presentation slide, `find` returns the picture as an `mlreportgen.ppt.TemplatePicture` object.

```
slide1 = ppt.Children(1);
templatePictureObj = find(slide1, "Picture")
```

```
templatePictureObj =
    TemplatePicture with properties:
```

```
XMLMarkup: '<p:pic><p:nvPicPr><p:cNvPr id="8" name="Picture"/><p:cNvPicPr><a:picLocks noChan
    Name: 'Picture'
      X: []
      Y: []
    Width: []
    Height: []
    Style: []
    Children: []
    Parent: [1x1 mlreportgen.ppt.Slide]
    Tag: 'ppt.TemplatePicture:1360:550'
    Id: '1360:550'
```

Replace the picture with a different picture.

```
street2 = Picture("street2.jpg");
replace(templatePictureObj, street2);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

## See Also

`mlreportgen.ppt.Presentation` | `mlreportgen.ppt.Slide` | `mlreportgen.ppt.Picture` | `mlreportgen.ppt.PicturePlaceholder`

## Topics

“Access PowerPoint Template Elements” on page 14-32

“Add and Replace Presentation Content” on page 14-58

## Introduced in R2019b

## replace

**Class:** `mlreportgen.ppt.TemplateTable`

**Package:** `mlreportgen.ppt`

Replace template table with another table

### Syntax

```
newTable = replace(templateTable, replacementTable)
```

### Description

`newTable = replace(templateTable, replacementTable)` replaces a template table with another table.

### Input Arguments

**templateTable — Template table to replace**

`mlreportgen.ppt.TemplateTable` object

Template table to replace, specified as an `mlreportgen.ppt.TemplateTable` object.

**replacementTable — Replacement table**

`mlreportgen.ppt.Table` object

Replacement table, specified as an `mlreportgen.ppt.Table` object.

### Output Arguments

**newTable — New table**

`mlreportgen.ppt.Table` object

New table in the presentation slide, specified as an `mlreportgen.ppt.Table` object.

## Examples

### Replace Template Table

Generate a presentation, `MyTablePresentation`, that you then use as the template presentation for another presentation. `MyTablePresentation` has one slide with one table

```
import mlreportgen.ppt.*
ppt = Presentation("MyTablePresentation");
open(ppt);
slidel = add(ppt, "Title and Table");
replace(slidel, "Title", "Magic Square Slide 1");
replace(slidel, "Table", Table(magic(3)));
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Create a presentation, `MyNewTablePresentation`, from `MyTablePresentation`. `MyTablePresentation` is the template presentation for `MyNewTablePresentation`,

```
ppt = Presentation("MyNewTablePresentation", "MyTablePresentation");
open(ppt);
```

Find the template table by using the `find` method of the slide object. Because the table comes from a template presentation slide, `find` returns the table as an `mlreportgen.ppt.TemplateTable` object.

```
slide1 = ppt.Children(1);
templateTableObj1 = find(slide1, "Table")
```

```
templateTableObj1 =
    TemplateTable with properties:
```

```
XMLMarkup: '<p:graphicFrame><p:nvGraphicFramePr><p:cNvPr id="3" name="Table"/><p:cNvGraphicF
Name: 'Table'
X: '838200emu'
Y: '1825625emu'
Width: '10515600emu'
Height: '4351338emu'
Style: []
Children: []
Parent: [1x1 mlreportgen.ppt.Slide]
Tag: 'ppt.TemplateTable:883:428'
Id: '883:428'
```

Replace the table on the slide with a table for a 4-by-4 magic square.

```
replace(templateTableObj1, Table(magic(4)));
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

## See Also

`mlreportgen.ppt.Presentation` | `mlreportgen.ppt.Slide` | `mlreportgen.ppt.Table` | `mlreportgen.ppt.TablePlaceholder`

## Topics

“Access PowerPoint Template Elements” on page 14-32

“Add and Replace Presentation Content” on page 14-58

## Introduced in R2019b

## add

**Class:** mlreportgen.ppt.TextBox

**Package:** mlreportgen.ppt

Add paragraph to text box

### Syntax

```
paraObj = add(textBox, content)
add(textBox, contents)
```

### Description

paraObj = add(textBox, content) adds a paragraph to a text box.

add(textBox, contents) adds multiple paragraphs to a text box .

### Examples

#### Add Text to Text Box

Create a presentation with one slide.

```
import mlreportgen.ppt.*
ppt = Presentation('myTextBoxAddPresentation.pptx');
slide = add(ppt, 'Blank');
```

Add a text box to the blank slide.

```
tb = TextBox();
tb.X = '1in';
tb.Y = '1in';
tb.Width = '4 in';
tb.Height = '2in';
```

```
add(slide, tb);
```

Add text to the text box.

```
para = add(tb, 'This is the content');
para.Bold = true;
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

### Input Arguments

**textBox** — Text box to add text to  
mlreportgen.ppt.TextBox object

Text box to add text to, specified as an `mlreportgen.ppt.TextBox` object.

**content — Text to add to text box**

character vector | string scalar | `mlreportgen.ppt.Paragraph`

Content to add to a text box, specified as a character vector, string scalar, or `mlreportgen.ppt.Paragraph` object.

**contents — Multiple paragraphs to add**

string array | cell array

Multiple paragraphs to add, specified as a string array or as a cell array that can contain character vectors, `mlreportgen.ppt.Paragraph` objects, or combination of character vectors, strings scalars, string arrays, or `mlreportgen.ppt.Paragraph` objects. For a cell array, contents of inner cell arrays or string arrays are indented from the contents of the outer cell array.

## Output Arguments

**paraObj — Paragraph**

`mlreportgen.ppt.Paragraph` object

Paragraph, returned as an `mlreportgen.ppt.Paragraph` object.

## See Also

`mlreportgen.ppt.TextBox` | `replace` | `mlreportgen.ppt.Paragraph`

### Topics

“Add and Replace Text” on page 14-59

“Create and Format Text” on page 14-65

### Introduced in R2015b

## replace

**Class:** mlreportgen.ppt.TextBox

**Package:** mlreportgen.ppt

Replace text box paragraphs

### Syntax

```
paraObj = replace(textBox, content)
replace(textBox, contents)
```

### Description

paraObj = replace(textBox, content) replaces a paragraph in a text box.

replace(textBox, contents) replaces multiple paragraphs in a text box placeholder.

### Examples

#### Replace Text in a Text Box

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation('myTextBoxReplacePresentation.pptx');
slide = add(ppt, 'Blank');
```

Create an mlreportgen.ppt.Paragraph object.

```
p = Paragraph('Hello World');
```

Add a text box to the blank slide (slide).

```
tb = TextBox();
tb.X = '1in';
tb.Y = '1in';
tb.Width = '4 in';
tb.Height = '2in';
```

```
add(slide, tb);
```

Add the paragraph to the text box.

```
add(tb, p);
```

Replace the content of the text box.

```
replace(tb, 'This is the real content');
```

Close and view the presentation.



```
close(ppt);  
rptview(ppt);
```

## Input Arguments

### **textBox** — Text box to replace text in

`mlreportgen.ppt.TextBox` object

Text box replace text in, specified as an `mlreportgen.ppt.TextBox` object.

### **content** — Text to use as replacement

character vector | string scalar | `mlreportgen.ppt.Paragraph`

Text to use as replacement, specified as a character vector, string scalar, or `mlreportgen.ppt.Paragraph` object.

### **contents** — Multiple paragraphs to use as replacement

string array | cell array

Multiple paragraphs to use as replacement, specified as a string array or as a cell array that can contain character vectors, `mlreportgen.ppt.Paragraph` objects, or combination of character vectors, strings scalars, string arrays, or `mlreportgen.ppt.Paragraph` objects. For a cell array, contents of inner cell arrays or string arrays are indented from the contents of the outer cell array.

## Output Arguments

### **paraObj** — Paragraph

`mlreportgen.ppt.Paragraph` object

Paragraph, returned as an `mlreportgen.ppt.Paragraph` object.

## See Also

`mlreportgen.ppt.TextBox` | `add` | `mlreportgen.ppt.Paragraph`

### Topics

“Add and Replace Text” on page 14-59

“Create and Format Text” on page 14-65

### Introduced in R2015b

## add

**Class:** mlreportgen.ppt.TextBoxPlaceholder

**Package:** mlreportgen.ppt

Add content to text box placeholder

### Syntax

```
addedParagraphObj = add(textBoxPlaceholder,paragraph)
```

### Description

`addedParagraphObj = add(textBoxPlaceholder,paragraph)` adds text as a paragraph in the specified text box placeholder and returns the paragraph object.

### Examples

#### Add Paragraph to Slide Title

To add a paragraph to a slide title, use the `add` method of the `TextBoxPlaceholder` object that represents the title. This example creates a presentation with a `Title` and `Table` slide and then creates a second presentation from the first presentation. In the second presentation, the example adds content to the title of the `Title` and `Table` slide.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation and add a `Title` and `Table` slide to the presentation.

```
ppt = Presentation("myPPT1.pptx");  
open(ppt);  
slide = add(ppt,"Title and Table");
```

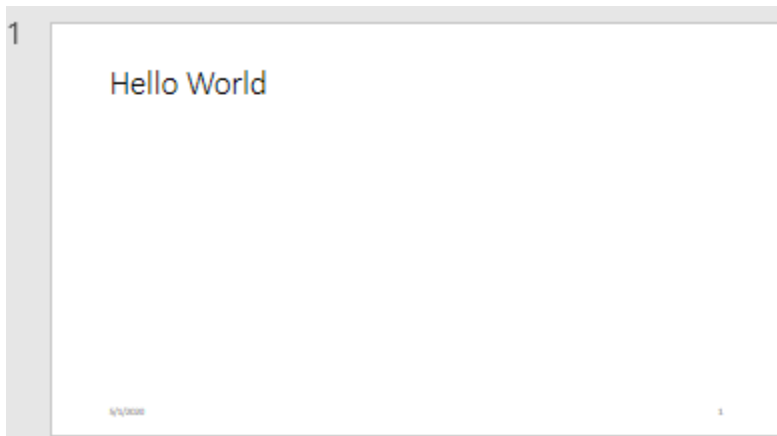
Find the text box placeholder for the title and replace the content.

```
contents = find(slide,"Title");  
replace(contents(1),"Hello World");
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

Here is the title slide in the generated presentation:



Create a second presentation, using the first presentation as the template.

```
ppt = Presentation("myPPT2.pptx", "myPPT1.pptx");
open(ppt);
```

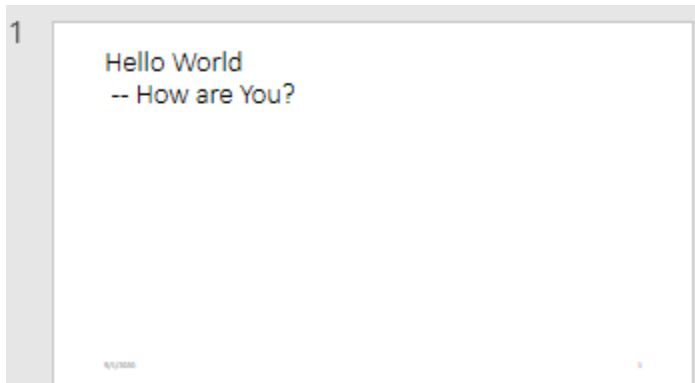
Find the text box placeholder for the title and add content to the title.

```
contents = find(ppt, "Title");
add(contents(1), "-- How are You?");
```

Close and view the presentation.

```
close(ppt);
rptview(ppt)
```

In the second presentation, the title has the content from the template presentation and the added content.



## Input Arguments

### **textBoxPlaceholder** — Text box placeholder

mlreportgen.ppt.TextBoxPlaceholder object

Text box placeholder, specified as an mlreportgen.ppt.TextBoxPlaceholder object.

**paragraph — Paragraph to add to text box placeholder**

character vector | string scalar | mlreportgen.ppt.Paragraph object

Paragraph to add to the text box placeholder, specified as a character vector, string scalar, or mlreportgen.ppt.Paragraph object.

**Output Arguments****addedParagraphObj — Added paragraph**

mlreportgen.ppt.Paragraph object

Added paragraph, returned as an mlreportgen.ppt.Paragraph object.

**See Also**

mlreportgen.ppt.TextBoxPlaceholder | replace | mlreportgen.ppt.Paragraph

**Topics**

“Access PowerPoint Template Elements” on page 14-32

“Add and Replace Text” on page 14-59

**Introduced in R2015b**

# replace

**Class:** mlreportgen.ppt.TextBoxPlaceholder

**Package:** mlreportgen.ppt

Replace text box placeholder content

## Syntax

```
replacementParagraphObj = replace(textBoxPlaceholder, paragraph)
```

## Description

`replacementParagraphObj = replace(textBoxPlaceholder, paragraph)` replaces the content in the specified text box placeholder with a paragraph and returns the paragraph object.

## Examples

### Replace Title Text in Placeholder

Add a title slide to a presentation and then replace the slide placeholders for the title and subtitle with your title and subtitle text.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation.

```
ppt = Presentation("myTextBoxPlaceholderPresentation.pptx");
open(ppt);
```

Add a title slide.

```
slide = add(ppt, "Title Slide");
```

Use the `find` method of the slide object to find the placeholder object that has the name `Title`.

```
titlePlaceholderObj = find(slide, "Title")
```

```
titlePlaceholderObj =
  TextBoxPlaceholder with properties:
```

```

        Bold: []
        Font: []
ComplexScriptFont: []
        FontColor: []
        FontSize: []
        Italic: []
        Strike: []
        Subscript: []
```

```
Superscript: []
Underline: []
BackgroundColor: []
VAlign: []
Name: 'Title'
X: '1524000emu'
Y: '1122363emu'
Width: '9144000emu'
Height: '2387600emu'
Style: []
Children: []
Parent: [1x1 mlreportgen.ppt.Slide]
Tag: 'ppt.TextBoxPlaceholder:21:61'
Id: '21:61'
```

The find method returns an `mlreportgen.ppt.TextBoxPlaceholder` object.

Replace the placeholder content with the title text.

```
replace(titlePlaceholderObj(1), "My Title");
```

Find the placeholder object for the subtitle.

```
subtitlePlaceholderObj = find(slide, "Subtitle")
```

```
subtitlePlaceholderObj =
  TextBoxPlaceholder with properties:
```

```
      Bold: []
      Font: []
ComplexScriptFont: []
      FontColor: []
      FontSize: []
      Italic: []
      Strike: []
      Subscript: []
      Superscript: []
      Underline: []
BackgroundColor: []
VAlign: []
Name: 'Subtitle'
X: '1524000emu'
Y: '3602038emu'
Width: '9144000emu'
Height: '1655762emu'
Style: []
Children: []
Parent: [1x1 mlreportgen.ppt.Slide]
Tag: 'ppt.TextBoxPlaceholder:22:62'
Id: '22:62'
```

The placeholder object for the subtitle is also an `mlreportgen.ppt.TextBoxPlaceholder` object.

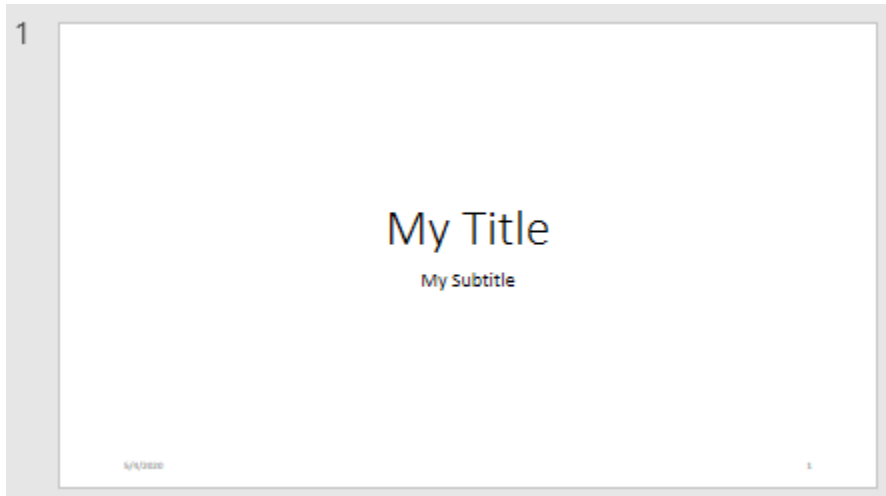
Replace the placeholder content with the text for the subtitle.

```
replace(subtitlePlaceholderObj(1), "My Subtitle");
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here is the title slide in the generated presentation:



## Input Arguments

### **textBoxPlaceholder** — Text box placeholder

`mlreportgen.ppt.TextBoxPlaceholder` object

Text box placeholder, specified as an `mlreportgen.ppt.TextBoxPlaceholder` object.

### **paragraph** — Paragraph to use as replacement

character vector | string scalar | `mlreportgen.ppt.Paragraph` object

Paragraph to use as the replacement for the text box placeholder content, specified as a character vector, string scalar, or `mlreportgen.ppt.Paragraph` object.

## Output Arguments

### **replacementParagraphObj** — Replacement paragraph

`mlreportgen.ppt.Paragraph` object

Replacement paragraph, returned as an `mlreportgen.ppt.Paragraph` object.

## See Also

`mlreportgen.ppt.TextBoxPlaceholder` | `add` | `mlreportgen.ppt.Paragraph`

## Topics

“Access PowerPoint Template Elements” on page 14-32

“Add and Replace Text” on page 14-59

**Introduced in R2015b**



# mlreportgen.finder.AxesResult.getReporter

**Class:** mlreportgen.finder.AxesResult

**Package:** mlreportgen.finder

Get axes reporter

## Syntax

```
reporter = getReporter(axesFinderResult)
```

## Description

`reporter = getReporter(axesFinderResult)` returns the axes reporter associated with the specified `mlreportgen.finder.AxesResult` object.

## Input Arguments

**axesFinderResult** — Axes finder results

mlreportgen.finder.AxesResult object

Axes finder results, specified as an `mlreportgen.finder.AxesResult` object.

## Output Arguments

**reporter** — Axes reporter

mlreportgen.report.Axes object

Axes reporter, returned as an `mlreportgen.report.Axes` object.

## See Also

mlreportgen.finder.AxesFinder | mlreportgen.finder.AxesResult |  
mlreportgen.report.Axes

**Introduced in R2021b**

## getPropertyValues

**Class:** `mlreportgen.finder.MATLABVariableResult`

**Package:** `mlreportgen.finder`

Get property values of MATLAB variable search result object

### Syntax

```
propVals = getPropertyValues(resObj,propNames)
```

### Description

`propVals = getPropertyValues(resObj,propNames)` returns the values of the properties specified by `propNames` in the MATLAB variable search result object, `resObj`. The specified properties can belong to the result object or to the variable represented by the result object.

### Input Arguments

**resObj — MATLAB variable search result object**

`mlreportgen.finder.MATLABVariableResult` object

MATLAB variable search result, specified as an `mlreportgen.finder.MATLABVariableResult` object.

**propNames — List of requested properties**

string array | cell array of strings

List of requested properties, specified as a string array or a cell array of strings. Valid values for elements in the array are, in any order:

Requested Property	Returned Value
"Name"	Name of the variable, specified as a string scalar
"Class"	Data type of the variable, specified as a character array
"Size"	Dimensions of the variable, specified as a double array
"Bytes"	Number of bytes used for storing the variable in the computer memory, specified as a double scalar
"Sparse"	Whether the variable is a sparse matrix, specified as a logical
"Complex"	Whether the variable is a complex number, specified as a logical
"Value"	Value of the variable

Requested Property	Returned Value
"Global"	Whether the variable is global, specified as a logical
The name of any property of resObj	The value of the property
The name of any property of resObj that can be accessed by dot notation. <b>Note:</b> This option only applies to variables of type struct or to variables that are instances of classes.	The value of the property

**Note** If the variable represented by resObj has a property with the same name as a property of resObj, the method getPropertyValues retrieves the property value of resObj. For example:

- 1 In the base workspace, define a struct with a field named Place and a field named Location.

```
myStruct.Place = "USA";
myStruct.Location = "America";
```

- 2 Use the find method to create a MATLABVariableResult object for the structure myStruct.

```
resObj = find(mlreportgen.finder.MATLABVariableFinder(...
    Container="MATLAB",Name="myStruct"));
```

- 3 Use the getPropertyValues method to get the values of the Place and Location properties.

```
getPropertyValues(resObj,["Place","Location"])
```

The value of the Location property is "MATLAB" from resObj.

```
ans =
```

```
1x2 cell array
```

```
 {"USA"} {"MATLAB"}
```

## Output Arguments

### propVals — Values of specified properties

cell array

Values of the specified properties, returned as a cell array.

## See Also

mlreportgen.report.SummaryTable

**Introduced in R2022a**

## mlreportgen.report.Axes.createTemplate

**Class:** mlreportgen.report.Axes

**Package:** mlreportgen.report

Create axes reporter template

### Syntax

```
template = mlreportgen.report.Axes.createTemplate(templatePath,type)
```

### Description

`template = mlreportgen.report.Axes.createTemplate(templatePath,type)` creates a copy of the `mlreportgen.report.Axes` template for the report type specified by `type` at the location specified by `templatePath`. Use the copied template as a starting point to design a custom reporter template for your report.

### Input Arguments

**templatePath** — Path and file name of new template

character vector | string scalar

Path and file name of the new template, specified as a character vector or a string scalar.

**type** — Type of template

'html' | 'html-file' | 'docx' | 'pdf'

Type of template, specified as 'html', 'html-file', 'docx', or 'pdf'.

### Output Arguments

**template** — Path and file name of template copy

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

### Examples

#### Create an Axes Reporter Template

Create a copy of the HTML template for the `mlreprtgen.report.Axes` class and save it with the name `myTemplate` in the `mytemplates` folder.

```
template = mlreportgen.report.Axes.createTemplate('mytemplates/myTemplate','html');
```

After you modify the template, use it by setting the `TemplateSrc` property of an axes reporter to the path of the template file.

**See Also**

`mlreportgen.finder.AxesFinder` | `mlreportgen.report.Axes` |  
`mlreportgen.report.Figure`

**Introduced in R2021b**

## mlreportgen.report.Axes.customizeReporter

**Class:** mlreportgen.report.Axes

**Package:** mlreportgen.report

Create class derived from axes reporter class

### Syntax

```
reporter = mlreportgen.report.Axes.customizeReporter(classPath)
```

### Description

`reporter = mlreportgen.report.Axes.customizeReporter(classPath)` creates a reporter class definition file that defines a subclass of `mlreportgen.report.Axes` at the location specified by `classPath`. The method also copies the default reporter templates to the `resources/templates` subfolder of the folder that contains the class definition file. Use the class definition file as a starting point to design a custom reporter class.

### Input Arguments

#### **classPath — Path and name of new class definition file**

string scalar | character vector

Path and name of a new class definition file, specified as a string scalar or a character vector.

Specify a relative path or an absolute path. For example, this code creates `myClass.m` in the subfolder `myFolder` of the current folder.

```
mlreportgen.report.Axes.customizeReporter("myFolder/myClass")
```

To create the reporter class in a class folder, precede the class name with the `@` character. Do not specify the `.m` extension. For example, this code creates `myClass.m` in the subfolder `myFolder/@myClass` in the current folder.

```
mlreportgen.report.Axes.customizeReporter("myFolder/@myClass")
```

See “Folders Containing Class Definitions”.

To create the reporter class in a package, precede the folder name with the `+` character. For example, this code creates a reporter in the `myOrg` package folder in the current folder.

```
mlreportgen.report.Axes.customizeReporter("+myOrg/@myClass")
```

### Output Arguments

#### **reporter — Path and file name of new reporter class**

string scalar

Path and file name of the new reporter class, returned as a string scalar.

## Examples

### Create Custom Axes Reporter

Create a custom reporter and its associated default templates. The derived class file is created at the specified path relative to the current working folder. In this case, the path to the `myAxes.m` class file is `<current working folder>/newAxes/@myAxes/myAxes.m`. The default title page templates are in the `<current working folder>/newAxes/@myAxes/resources/templates` folder.

```
mlreportgen.report.Axes.customizeReporter("newAxes/@myAxes")
```

After editing this new class file, you can use it as your axes reporter.

```
axes = myAxes();
```

### See Also

`mlreportgen.finder.AxesFinder` | `mlreportgen.report.Axes` | `mlreportgen.report.Figure`

### Introduced in R2021b

## **mlreportgen.report.Axes.getClassFolder**

**Class:** mlreportgen.report.Axes

**Package:** mlreportgen.report

Get location of folder containing mlreportgen.report.Axes class definition file

### **Syntax**

```
path = mlreportgen.report.Axes.getClassFolder()
```

### **Description**

path = mlreportgen.report.Axes.getClassFolder() returns the path of the folder that contains the mlreportgen.report.Axes class definition file.

### **Output Arguments**

**path** — mlreportgen.report.Axes class definition file location

character vector

mlreportgen.report.Axes class definition file location, returned as a character vector.

### **Examples**

#### **Get Axes Reporter Class Folder**

Get the location of the folder that contains the mlreportgen.report.Axes class definition file.

```
path = mlreportgen.report.Axes.getClassFolder()
```

### **See Also**

mlreportgen.finder.AxesFinder | mlreportgen.report.Axes |  
mlreportgen.report.Figure

**Introduced in R2021b**



# getSnapshotImage

**Class:** mlreportgen.report.Axes

**Package:** mlreportgen.report

Get axes image path

## Syntax

```
axesPath = mlreportgen.report.Axes.getSnapshotImage(axesReporter, report)
```

## Description

`axesPath = mlreportgen.report.Axes.getSnapshotImage(axesReporter, report)` creates an image of the axes specified by `axesReporter` in the specified report and returns a path to a file containing the image. Use this method to customize the layout of axes in your report.

## Input Arguments

### axesReporter — Axes reporter

mlreportgen.report.Axes

Axes reporter, specified as an mlreportgen.report.Axes object.

### report — Report

report object

Report, specified as a report object. To create the report input, use mlreportgen.report.Report.

## Output Arguments

### axesPath — Path of axes image

string

Path of the axes image, returned as string.

## Examples

### Change the Size of a Snapshot Image

```
import mlreportgen.report.*
import mlreportgen.dom.*

% Create a PDF report
rpt = Report("Axes Image Size", "pdf");

% Create an axes object
ax = axes(Position=[0.1 0.1 0.7 0.7]);
```

```
x1 = linspace(0,10,100);  
y1 = sin(x1);  
plot(ax,x1,y1);  
  
% Create an axes reporter object for 'ax'  
axesRpt = Axes(ax);  
% Access the image path by using getSnapshotImage  
axesImg = Image(getSnapshotImage(axesRpt,rpt));  
axesImg.Width = "3in";  
axesImg.Height = "3in";  
  
append(rpt,axesImg)  
close(rpt);  
rptview(rpt);
```

### **See Also**

[mlreportgen.finder.AxesFinder](#) | [mlreportgen.report.Axes](#) | [mlreportgen.report.Figure](#)

**Introduced in R2021b**

# mlreportgen.report.BaseTable.createTemplate

**Class:** mlreportgen.report.BaseTable

**Package:** mlreportgen.report

Create table template

## Syntax

```
template = mlreportgen.report.BaseTable.createTemplate(templatePath,type)
```

## Description

`template = mlreportgen.report.BaseTable.createTemplate(templatePath,type)` creates a copy of the default table reporter template specified by `type` at the location specified by `templatePath`. You can use the copied template as a starting point to design a custom table template for your report.

## Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

"html" | "html-file" | "docx" | "pdf"

Type of template, specified as "html", "html-file", "docx", or "pdf".

## Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

## Examples

### Create Title Page Template

Before you run this example, create a copy of the default HTML BaseTable template in the `mytemplates` folder. Name the copied template `myTable.htm`. Edit the template as desired. To use the new template for the title page, assign its path to the BaseTable `TemplateSrc` property.

```
import mlreportgen.report.*
rpt = Report('My Report','html');
table = BaseTable(magic(5));
```

```
table.Title = 'Rank 5 Magic Square';  
template = BaseTable.createTemplate('mytemplates\myTable','html');  
table.TemplateSrc = template;
```

**See Also**

mlreportgen.report.BaseTable | mlreportgen.report.Report |  
mlreportgen.report.Reporter

**Topics**

“Modify Styles in a Microsoft Word Template” on page 13-126

“Modify Styles in HTML Templates” on page 13-137

“Modify Styles in PDF Templates” on page 13-138

**Introduced in R2017b**

# mlreportgen.report.BaseTable.customizeReporter

**Class:** mlreportgen.report.BaseTable

**Package:** mlreportgen.report

Create custom base table reporter class

## Syntax

```
reporter = mlreportgen.report.BaseTable.customizeReporter(classpath)
```

## Description

`reporter = mlreportgen.report.BaseTable.customizeReporter(classpath)` creates a base table class definition file that is a subclass of `mlreportgen.report.BaseTable`. The file is created at the specified `classpath` location. The `BaseTable.customizeReporter` method also copies the default base table templates to the `<classpath>/resources/template` folder. You can use the class definition file as a starting point to design a custom base table class for your report.

## Input Arguments

**classpath — Location of custom base table class**

current working folder (default) | string | character array

Location of custom base table class, specified as a string or character array. The `classpath` argument also supports specifying a folder with `@` before the class name.

## Output Arguments

**reporter — Base table reporter path**

string

Base table reporter path, returned as the string specifying the path to the derived report class file.

## Examples

### Create Custom Base Table Reporter

Create a custom base table reporter and its associated default templates. The derived class file is created at the specified path relative to the current working folder. In this case, the path to the `MyBaseTable.m` class file is `<current working folder>/newTable/@MyBaseTable/MyBaseTable.m`. The default base table templates are in the `<current working folder>/newTable/@MyMyBaseTable/resources/templates` folder.

```
import mlreportgen.report.*
BaseTable.customizeReporter('newTable/@MyBaseTable');
```

After editing this new class file, you can use it as your base table section reporter.

```
basetbl = MyBaseTable();
```

**See Also**

`mlreportgen.report.BaseTable` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Report`

**Introduced in R2017b**

# mlreportgen.report.BaseTable.getClassFolder

**Class:** mlreportgen.report.BaseTable

**Package:** mlreportgen.report

Base table class definition file location

## Syntax

```
path = mlreportgen.report.BaseTable.getClassFolder()
```

## Description

`path = mlreportgen.report.BaseTable.getClassFolder()` returns the path of the folder that contains the base table class definition file.

## Output Arguments

**path** — Base table class definition file location

character array

Base table class definition file location, returned as a character array.

## See Also

mlreportgen.report.BaseTable | mlreportgen.report.Report |

mlreportgen.report.Reporter

**Introduced in R2017b**

## getContentReporter

**Class:** `mlreportgen.report.BaseTable`

**Package:** `mlreportgen.report`

Get base table content hole reporter

### Syntax

```
reporter = getContentReporter(baseTable)
```

### Description

`reporter = getContentReporter(baseTable)` returns a hole reporter that the base table reporter uses to insert its content into a report. The default `BaseTableContent` template is in the template library of the `BaseTable` reporter. This template contains only a hole for the table generated from the `Content` property of the base table reporter. Use this method to customize the `Content` template to add a table to a report.

### Input Arguments

**baseTable** — Base table reporter object

reporter object

Base table reporter object, specified as an `mlreportgen.report.BaseTable` object.

### Output Arguments

**reporter** — Base table content hole reporter

reporter object

Base table content hole reporter, returned as a reporter object.

### Examples

#### Use Customized Table Content Template

Before you run this example, perform these steps.

- 1 Use the `mlreportgen.report.Report.createTemplate` method to create a copy of the `mlreportgen.report.Report` template and name it `'myreporttemplate.pdfxtx'`. Create the template in the same folder as the report.
- 2 Use the `mlreportgen.report.BaseTable.createTemplate` method to create a copy of the `BaseTable` template in the report folder.
- 3 Unzip the templates.
- 4 Copy the `BaseTableContent` template from the `BaseTable` template library to the template library of `myreporttemplate.pdfxtx`.



- 5 Copy the BaseTableContent style from the BaseTable style sheet to the myreporttemplate.pdfctx style sheet.
- 6 Edit the BaseTableContent template and style in the template library of myreporttemplate.pdfctx to meet your requirements.
- 7 Rezip the templates.
- 8 Delete the copy of the BaseTable template.

```
import mlreportgen.report.*
import mlreportgen.dom.*
rpt = Report('myreport', 'pdf', 'myreporttemplate');
tableRptr = BaseTable();
tableRptr.Content = Table(magic(5));
contentRptr = getContentReporter(tableRptr);
contentRptr.TemplateSrc = rpt;
tableRptr.Content = contentRptr;
add(rpt, tableRptr);
close(rpt);
```

## See Also

mlreportgen.report.BaseTable | mlreportgen.report.Report |  
mlreportgen.report.Reporter

**Introduced in R2017b**

## getTitleReporter

**Class:** `mlreportgen.report.BaseTable`

**Package:** `mlreportgen.report`

Get base table title reporter

### Syntax

```
reporter = getTitleReporter(table)
```

### Description

`reporter = getTitleReporter(table)` returns a reporter that the `BaseTable` reporter (`table`) uses to format the value specified by its `Title` property. Use `getTitleReporter` to customize the title alignment, position, and appearance.

### Input Arguments

**table** — Table for the report

base table reporter

Table for the report, specified as a base table reporter object. To create the `table` input, use `mlreportgen.report.BaseTable`.

### Output Arguments

**reporter** — Base table title reporter

reporter object

Base table title reporter, returned as a reporter object.

### Examples

#### Use Customized Table Title Style

Create a style for the title of your table that differs from the default style. First, create a custom `BaseTable` template using these steps:

- 1 Create a copy of the PDF template of the reporter using the `createTemplate` method of the `BaseTable` reporter. Name the copy `MyTableTemplate.pdf` and store it in the report folder.
- 2 Unzip the template.
- 3 Open the style sheet file of the template in a text editor.
- 4 Edit the `BaseTableTitle` style in the style sheet file of the template so it meets your requirements.
- 5 Save the style sheet file.
- 6 Rezip the template.

Then, use the getTitleReporter method and the TemplateSrc property to use your template.

```
.  
import mlreportgen.report.*  
import mlreportgen.dom.*  
rpt = Report('MyReport', 'pdf');  
table = BaseTable();  
table.Title = 'My Table';  
titleReporter = getTitleReporter(table);  
titleReporter.TemplateSrc = 'MyTableTemplate.pdf';  
table.Title = titleReporter;
```

## See Also

[mlreportgen.report.BaseTable](#) | [mlreportgen.report.Report](#) | [mlreportgen.report.Reporter](#)

## Topics

[“Modify Styles in a Microsoft Word Template” on page 13-126](#)

[“Modify Styles in HTML Templates” on page 13-137](#)

[“Modify Styles in PDF Templates” on page 13-138](#)

## Introduced in R2017b

## mlreportgen.report.Equation.createTemplate

**Class:** mlreportgen.report.Equation

**Package:** mlreportgen.report

Create equation reporter template

### Syntax

```
template = mlreportgen.report.Equation.createTemplate(templatePath,type)
```

### Description

`template = mlreportgen.report.Equation.createTemplate(templatePath,type)` creates a copy of the equation reporter template specified by `type` at the location specified by `templatePath`. You can use the copied template as a starting point to design a custom equation template for your report.

### Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

"html" | "html-file" | "docx" | "pdf"

Type of template, specified as "html", "html-file", "docx", or "pdf".

### Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

### Examples

#### Create Equation Template

Create a copy of the HTML template for the equation reporter and save it with the name `myEquationTemplate` in the `mytemplates` folder.

```
template = mlreportgen.report.Equation.createTemplate...  
('mytemplates/myEquationTemplate','html');
```

After you modify the template, you can use it by setting the `TemplateSrc` property of the reporter.

```
rptr = mlreportgen.report.Equation;  
rptr.TemplateSrc = template;
```

## See Also

[mlreportgen.report.Equation](#) | [mlreportgen.report.Report](#) |  
[mlreportgen.report.Reporter](#)

## Topics

“Modify Styles in a Microsoft Word Template” on page 13-126

“Modify Styles in HTML Templates” on page 13-137

“Modify Styles in PDF Templates” on page 13-138

## Introduced in R2017b

## mlreportgen.report.Equation.customizeReporter

**Class:** mlreportgen.report.Equation

**Package:** mlreportgen.report

Create custom equation reporter class

### Syntax

```
reporter = mlreportgen.report.Equation.customizeReporter(classpath)
```

### Description

`reporter = mlreportgen.report.Equation.customizeReporter(classpath)` creates an equation class definition file that is a subclass of `mlreportgen.report.Equation`. The file is created at the specified `classpath` location. The `Equation.customizeReporter` method also copies the default title page templates to the `<classpath>/resources/template` folder. You can use the new class definition file as a starting point to design a custom equation class for your report.

For example:

```
mlreportgen.report.Equation.customizeReporter("path_folder/MyClassA.m")
mlreportgen.report.Equation.customizeReporter("+package/@MyClassB")
```

### Input Arguments

**classpath — Location of custom equation class**

current working folder (default) | string | character array

Location of custom equation class, specified as a string or character array. The `classpath` argument also supports specifying a folder with `@` before the class name.

### Output Arguments

**reporter — Equation reporter path**

string

Equation reporter path, returned as the string specifying the path to the derived report class file.

### Examples

#### Create Custom Equation Reporter

Create a custom equation reporter and its associated default template. The derived class file is created at the specified path relative to the current working folder. In this case, the path to the `MyEq.m` class file is `<current working folder>/newEq/@MyEq/MyEq.m`. The default title page templates are in the `<current working folder>/newEq/@MyEq/resources/templates` folder.

```
import mlreportgen.report.*
Equation.customizeReporter('newEq/@MyEq');
```

After editing this new class file, you can use it as your title page reporter.

```
eq = MyEq();
```

### **See Also**

[mlreportgen.report.Equation](#) | [mlreportgen.report.Report](#) | [mlreportgen.report.Reporter](#)

**Introduced in R2017b**

## **mlreportgen.report.Equation.getClassFolder**

**Class:** mlreportgen.report.Equation

**Package:** mlreportgen.report

Equation class definition file location

### **Syntax**

```
path = mlreportgen.report.Equation.getClassFolder()
```

### **Description**

`path = mlreportgen.report.Equation.getClassFolder()` returns the path of the folder that contains the equation class definition file.

### **Output Arguments**

**path — Equation class definition file location**

character array

Equation class definition file location, returned as a character array.

### **See Also**

`mlreportgen.report.Equation` | `mlreportgen.report.Report` |

`mlreportgen.report.Reporter`

**Introduced in R2017b**



# getContentReporter

**Class:** `mlreportgen.report.Equation`

**Package:** `mlreportgen.report`

Get equation content hole reporter

## Syntax

```
reporter = getContentReporter(equation, report)
```

## Description

`reporter = getContentReporter(equation, report)` returns a hole reporter used to fill the Content hole in the template of the Equation reporter. The hole reporter contains an image of the formatted equation. This equation is generated from the LaTeX markup specified in `Content` property of the Equation reporter. Use this `getContentReporter` method to override the format specified by the Content hole reporter.

## Input Arguments

### **equation** — Equation reporter object

reporter object

Equation reporter object, specified as an `mlreportgen.report.Equation` object.

### **report** — Report

report object

Report, specified as a report object. To create the report input, use `mlreportgen.report.Report`.

## Output Arguments

### **reporter** — Equation content hole reporter

reporter object

Equation content hole reporter, returned as a reporter object.

## See Also

`mlreportgen.report.Equation` | `mlreportgen.report.Report` | `mlreportgen.report.Reporter`

**Introduced in R2017b**

## getSnapshotImage

**Class:** `mlreportgen.report.Equation`

**Package:** `mlreportgen.report`

Create equation image and return file path or data URL

### Syntax

```
equationImage = getSnapshotImage(equation,report)
```

### Description

`equationImage = getSnapshotImage(equation,report)` creates an image of the formatted equation and returns the path of the file or data URL that contains the image. The returned value depends on the value of the `UseDirectRenderer` property of the `mlreportgen.report.Equation` object specified by `equation`.

### Input Arguments

**equation — Equation reporter object**

`mlreportgen.report.Equation` object

Equation reporter object, specified as an `mlreportgen.report.Equation` object.

**report — Report**

report object

Report, specified as a report object. To create the report input, use `mlreportgen.report.Report`.

### Output Arguments

**equationImage — Equation image file path or data URL**

string scalar

Equation image file path or data URL, returned as a string scalar. If the `UseDirectRenderer` property of the `mlreportgen.report.Equation` object specified by the `equation` argument is `true`, `equationImage` is a data URL that contains an image of the equation. If the `UseDirectRenderer` property is `false`, `equationImage` is the path of the file that contains an image of the equation. Specify the image format by using the `SnapshotFormat` property of the `Equation` object.

### Examples

**Create a Left-Aligned, Numbered Equation**

```
import mlreportgen.report.*
import mlreportgen.dom.*
rpt = Report('equation', 'html');
```

```
ch = Chapter('Title', 'Equation');
eq = Equation('\int_{0}^{2} x^2 \sin(x) dx');
eq.FontSize = 12;
p = Paragraph('Eq 1: ');
p.Bold = true;
eqImg = Image(getSnapshotImage(eq, rpt));
t = Table({p, eqImg});
add(ch, t);
add(rpt, ch);
close(rpt);
rptview(rpt);
```

## See Also

`mlreportgen.report.Equation` | `mlreportgen.report.Report`

**Introduced in R2017b**

## mlreportgen.report.Figure.createTemplate

**Class:** mlreportgen.report.Figure

**Package:** mlreportgen.report

Create figure template

### Syntax

```
template = mlreportgen.report.Figure.createTemplate(templatePath,type)
```

### Description

`template = mlreportgen.report.Figure.createTemplate(templatePath,type)` creates a copy of the figure reporter template specified by `type` at the `templatePath` location. You can use the copied template as a starting point to design a custom figure template for your report.

### Input Arguments

**templatePath** — Path and file name of new template

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type** — Type of template

"html" | "html-file" | "docx" | "pdf"

Type of template, specified as "html", "html-file", "docx", or "pdf".

### Output Arguments

**template** — Path and file name of template copy

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

### Examples

#### Create Figure Template

Before you run this example, create a copy of the default HTML Figure template in the `mytemplates` folder. Name the copied template `myFigure.htm`. Edit the template as desired. To use the new template for the figure, assign its path to the `TemplateSrc` property of `mlreportgen.report.Figure`.

```
import mlreportgen.report.*
rpt = Report('My Report','html');
fig = Figure();
```

```
template = Figure.createTemplate('mytemplates\myFigure', 'html');  
fig.TemplateSrc = template;
```

## See Also

mlreportgen.report.Figure | mlreportgen.report.Reporter |  
mlreportgen.report.Report

## Topics

“Modify Styles in a Microsoft Word Template” on page 13-126

“Modify Styles in HTML Templates” on page 13-137

“Modify Styles in PDF Templates” on page 13-138

## Introduced in R2017b

## mlreportgen.report.Figure.customizeReporter

**Class:** mlreportgen.report.Figure

**Package:** mlreportgen.report

Create custom figure reporter class

### Syntax

```
reporter = mlreportgen.report.Figure.customizeReporter(classpath)
```

### Description

`reporter = mlreportgen.report.Figure.customizeReporter(classpath)` creates a figure class definition file that is a subclass of `mlreportgen.report.Figure`. The file is created at the specified `classpath` location. The `Figure.customizeReporter` method also copies the default figure templates to the `<classpath>/resources/template` folder. You can use the class definition file as a starting point to design a custom figure class for your report.

### Input Arguments

**classpath — Location of custom figure class**

current working folder (default) | string | character array

Location of custom figure class, specified as a string or character array. The `classpath` argument also supports specifying a folder with `@` before the class name.

### Output Arguments

**reporter — Figure reporter path**

string

Figure reporter path, returned as the string specifying the path to the derived report class file.

### Examples

#### Create Custom Figure Reporter

Create a custom figure reporter and its associated default templates. The derived class file is created at the specified path relative to the current working folder. In this case, the path to the `MyFigure.m` class file is `<current working folder>/newImage/@MyFigure/myFigure.m`. The default title page templates are in the `<current working folder>/newImage/@MyFigure/resources/templates` folder.

```
import mlreportgen.report.*  
Figure.customizeReporter('newImage/@MyFigure');
```

After editing this new class file, you can use it as your Figure section reporter.

```
fig = MyFigure();
```

## See Also

[mlreportgen.report.Figure](#) | [mlreportgen.report.Reporter](#) |  
[mlreportgen.report.Report](#)

**Introduced in R2017b**

## **mlreportgen.report.Figure.getClassFolder**

**Class:** mlreportgen.report.Figure

**Package:** mlreportgen.report

Figure class definition file location

### **Syntax**

```
path = mlreportgen.report.Figure.getClassFolder()
```

### **Description**

`path = mlreportgen.report.Figure.getClassFolder()` returns the path of the folder that contains the figure class definition file.

### **Output Arguments**

**path** — **Figure class definition file location**

character array

Figure class definition file location, returned as a character array.

### **See Also**

mlreportgen.report.Figure | mlreportgen.report.Reporter |

mlreportgen.report.Report

**Introduced in R2017b**



# getSnapshotImage

**Class:** mlreportgen.report.Figure

**Package:** mlreportgen.report

Get snapshot image path

## Syntax

```
imgpath = getSnapshotImage(figReporter,report)
```

## Description

`imgpath = getSnapshotImage(figReporter,report)` creates an image of the figure window specified by `figReporter` and returns a path to a file containing the image. Use this method to customize the layout of figures in your report.

## Input Arguments

**figReporter** — Figure reporter object

reporter object

Figure reporter object, specified as an `mlreportgen.report.Figure` object.

**report** — Report

report object

Report, specified as a report object. To create the report input, use `mlreportgen.report.Report`.

## Output Arguments

**imgpath** — Figure path

string

Figure path of file containing the figure, returned as a string.

## Examples

### Change the Size of a Snapshot Image

```
import mlreportgen.report.*
import mlreportgen.dom.*

rpt = Report('peaks');
surf(peaks(20));
figure = Figure();
peaks20 = Image(getSnapshotImage(figure,rpt));
peaks20.Width = '3in';
peaks20.Height = [];
```

```
figure = peaks20;
```

```
delete(gcf);  
add(rpt,figure)  
close(rpt);  
rptview(rpt);
```

**See Also**

`mlreportgen.report.Figure` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Report`

**Introduced in R2017b**

# mlreportgen.report.FormalImage.createTemplate

**Class:** mlreportgen.report.FormalImage

**Package:** mlreportgen.report

Create formal image template

## Syntax

```
template = mlreportgen.report.FormalImage.createTemplate(templatePath,type)
```

## Description

`template = mlreportgen.report.FormalImage.createTemplate(templatePath,type)` creates a copy of the formal image reporter template specified by `type` at the `templatePath` location . You can use the copied template as a starting point to design a custom formal image template for your report.

## Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

"html" | "html-file" | "docx" | "pdf"

Type of template, specified as "html", "html-file", "docx", or "pdf".

## Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

## Examples

### Create Formal Image Template

Before you run this example, create a copy of the default HTML Formal Image template in the `mytemplates` folder. Name the copied template `myImage.htm`. Edit the template as desired. To use the new template for the image, assign its path to the `TemplateSrc` property of `mlreportgen.report.FormalImage`.

```
import mlreportgen.report.*
rpt = Report('My Report','html');
```

```
image = FormalImage();  
template = FormalImage.createTemplate('mytemplates\myImage', 'html');  
image.TemplateSrc = template;
```

**See Also**

mlreportgen.report.FormalImage | mlreportgen.report.Reporter |  
mlreportgen.report.Report

**Topics**

“Modify Styles in a Microsoft Word Template” on page 13-126

“Modify Styles in HTML Templates” on page 13-137

“Modify Styles in PDF Templates” on page 13-138

**Introduced in R2017b**

# mlreportgen.report.FormalImage.customizeReporter

**Class:** mlreportgen.report.FormalImage

**Package:** mlreportgen.report

Create custom formal image reporter class

## Syntax

```
reporter = mlreportgen.report.FormalImage.customizeReporter(classpath)
```

## Description

`reporter = mlreportgen.report.FormalImage.customizeReporter(classpath)` creates a formal image class definition file that is a subclass of `mlreportgen.report.FormalImage`. The file is created at the specified `classpath` location. The `FormalImage.customizeReporter` method also copies the default formal image templates to the `<classpath>/resources/template` folder. You can use the class definition file as a starting point to design a custom formal image class for your report.

## Input Arguments

**classpath** — Location of custom formal image class

current working folder (default) | string | character array

Location of custom formal image class, specified as a string or character array. The `classpath` argument also supports specifying a folder with `@` before the class name.

## Output Arguments

**reporter** — Formal image reporter path

string

Formal image reporter path, returned as the string specifying the path to the derived report class file.

## Examples

### Create Custom Formal Image Reporter

Create a custom formal image reporter and its associated default templates. The derived class file is created at the specified path relative to the current working folder. In this case, the path to the `MyImage.m` class file is `<current working folder>/newImage/@MyImage/myImage.m`. The default title page templates are in the `<current working folder>/newImage/@MyImage/resources/templates` folder.

```
import mlreportgen.report.*
FormalImage.customizeReporter('newImage/@MyImage');
```

After editing this new class file, you can use it as your Formal Image section reporter.

```
image = MyImage();
```

**See Also**

`mlreportgen.report.FormalImage` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Report`

**Introduced in R2017b**

# getCaptionReporter

**Class:** mlreportgen.report.FormalImage

**Package:** mlreportgen.report

Get image caption reporter

## Syntax

```
reporter = getCaptionReporter(image)
```

## Description

`reporter = getCaptionReporter(image)` returns a reporter that generates the formal image caption based on the `Caption` property of `mlreportgen.report.FormalImage`. The caption can be any MATLAB or DOM object that can be appended to a DOM Paragraph. The caption formats override any corresponding formats in the template. Use this `getCaptionReporter` method to override the default caption formats.

## Input Arguments

### **image** — Image source

character vector | character array | DOM Image object

Source of image to add to report, specified as a character vector or character array, or as a DOM Image object. If you use a character vector or character array, specify the system path to the image file.

## Output Arguments

### **reporter** — Formal image caption reporter

reporter object

Formal image caption reporter for the image, returned as a reporter object.

## Examples

### **Use Non-Default Caption Style**

Create a style for your caption that differs from the default style. Before you run this example, create a template named `MyCaptionTemplate` and customize its `FormalImageCaption` style. Then, use the `getCaptionReporter` method and the `TemplateSrc` property to use your template.

```
import mlreportgen.report.*
rpt = Report();
image = FormalImage();
captionReporter = getCaptionReporter(image);
image.TemplateSrc = 'MyCaptionTemplate';
add(rpt,image);
```

**See Also**

`mlreportgen.report.FormalImage` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Report`

**Introduced in R2017b**



# mlreportgen.report.FormalImage.getClassFolder

**Class:** mlreportgen.report.FormalImage

**Package:** mlreportgen.report

Formal image class definition file location

## Syntax

```
path = mlreportgen.report.FormalImage.getClassFolder()
```

## Description

`path = mlreportgen.report.FormalImage.getClassFolder()` returns the path of the folder that contains the formal image class definition file.

## Output Arguments

**path** — Formal image class definition file location

character array

Formal image class definition file location, returned as a character array.

## See Also

mlreportgen.report.FormalImage | mlreportgen.report.Reporter |

mlreportgen.report.Report

**Introduced in R2017b**

## getImageReporter

**Class:** mlreportgen.report.FormalImage

**Package:** mlreportgen.report

Get formal image reporter

### Syntax

```
reporter = getImageReporter(image)
```

### Description

`reporter = getImageReporter(image)` returns a reporter that generates a formal image based on the `Image` property of `mlreportgen.report.FormalImage`. The `Image` format overrides any corresponding formats in the image template. Use this `getImageReporter` method to override the default image template of the image.

### Input Arguments

**image — Image source**

character vector | character array | DOM Image object

Source of image to add to report, specified as a character vector or character array, or as a DOM Image object. If you use a character vector or character array, specify the system path to the image file.

### Output Arguments

**reporter — Formal image reporter**

reporter object

Formal image reporter for the image, returned as a reporter object.

### Examples

**Use Nondefault Image Style**

Create a style that differs from the default image style. Before you run this example, create a template named `MyImageTemplate` and customize its `FormalImageImage` style. Then, use the `getImageReporter` method and the `TemplateSrc` property to use your template.

```
import mlreportgen.report.*
rpt = Report();
image = FormalImage();
imageReporter = getImageReporter(image);
image.TemplateSrc = 'MyImageTemplate';
add(rpt,image);
```

## **See Also**

`mlreportgen.report.FormalImage` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Report`

**Introduced in R2017b**

## mlreportgen.report.HTMLModuleTabs.createTemplate

**Class:** mlreportgen.report.HTMLModuleTabs

**Package:** mlreportgen.report

Create a copy of the HTML module tabs reporter template

### Syntax

```
template = mlreportgen.report.HTMLModuleTabs.createTemplate(templatePath,  
type)
```

### Description

`template = mlreportgen.report.HTMLModuleTabs.createTemplate(templatePath, type)` creates a copy of the `mlreportgen.report.HTMLModuleTabs` reporter template specified by `type` at the `templatePath` location. You can use the copied template as a starting point to design a custom HTML module tabs reporter template for your report.

### Input Arguments

**templatePath** — Path and file name of new template

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type** — Type of template

'html' | 'html-file'

Format of the output, specified as 'html' or 'html-file'.

### Output Arguments

**template** — Path and file name of template copy

string scalar

Path and file name of the copy of the template, returned as a string scalar. If the input argument `type` is 'html', the file name extension of the template is `htmlx`. If `type` is 'html-file', the file name extension is `html`.

### Examples

#### Create an HTML Module Tabs Reporter Template

Create a copy of the HTML template for the `mlreportgen.report.HTMLModuleTabs` reporter and save it with the name `myTabsReporterTemplate` in the `mytemplates` folder.

```
template = mlreportgen.report.HTMLModuleTabs.createTemplate...  
    ('mytemplates/myTabsReporterTemplate', 'html');
```

After you modify the template, you can use it by setting the `TemplateSrc` property of the reporter.

## See Also

`mlreportgen.report.HTMLModuleTabs` | `mlreportgen.report.Report`

**Introduced in R2020a**

# mlreportgen.report.HTMLModuleTabs.customizeReporter

**Class:** mlreportgen.report.HTMLModuleTabs

**Package:** mlreportgen.report

Create a custom HTML module tabs reporter

## Syntax

```
reporter = mlreportgen.report.HTMLModuleTabs.customizeReporter(classpath)
```

## Description

`reporter = mlreportgen.report.HTMLModuleTabs.customizeReporter(classpath)` creates a reporter class definition file that defines a subclass of `mlreportgen.report.HTMLModuleTabs.customizeReporter`. The `mlreportgen.report.HTMLModuleTabs` method creates the reporter class definition file at the location specified by `classpath`. The method also copies the default reporter templates to the `resources/templates` subfolder of the folder that contains the class definition file. You can use the class definition file as a starting point to design a custom HTML module tabs reporter class for your report.

## Input Arguments

### **classpath** — Path and name of new class definition file

string scalar | character vector

Path and name of new class definition file, specified as a string scalar or character vector.

You can specify a relative path or an absolute path. For example, this code creates `MyClass.m` in the subfolder `myFolder` of the current folder.

```
mlreportgen.report.HTMLModuleTabs.customizeReporter("myFolder/MyClass")
```

To create the reporter class in a class folder, precede the class name with the `@` character. Do not specify the `.m` extension. For example, this code creates `MyClass.m` in the subfolder `myFolder/@MyClass` in the current folder.

```
mlreportgen.report.HTMLModuleTabs.customizeReporter("myFolder/@MyClass")
```

See “Folders Containing Class Definitions”.

To create the reporter class in a class package, precede the folder name with the `+` character. For example, this code creates an HTML module tabs reporter in the `myOrg` package folder in the current folder.

```
mlreportgen.report.HTMLModuleTabs.customizeReporter("+myOrg/@MyClass")
```

## Output Arguments

**reporter** — Path and file name of new HTML module tabs reporter class

string scalar

Path and file name of new HTML module tabs reporter class, returned as a string scalar.

## Examples

### Create Custom HTML Module Tabs Reporter

Create a custom HTML module tabs reporter, MyHTMLTabs, and the associated default templates in the subfolder MyFolder of the current working folder.

```
mlreportgen.report.HTMLModuleTabs.customizeReporter('MyFolder/MyHTMLTabs')
```

```
ans =
```

```
    "MyFolder\MyHTMLTabs.m"
```

### See Also

mlreportgen.report.HTMLModuleTabs | mlreportgen.report.Report

**Introduced in R2020a**

## **mlreportgen.report.HTMLModuleTabs.getClassFolder**

**Class:** mlreportgen.report.HTMLModuleTabs

**Package:** mlreportgen.report

HTML module tabs reporter class definition file location

### **Syntax**

```
path = mlreportgen.report.HTMLModuleTabs.getClassFolder()
```

### **Description**

path = mlreportgen.report.HTMLModuleTabs.getClassFolder() returns the path of the folder that contains the mlreportgen.report.HTMLModuleTabs class definition file.

### **Output Arguments**

**path** — Class definition file location

character vector

Class definition file location, returned as a character vector.

### **Examples**

#### **Get HTML Module Tabs Reporter Class Folder**

Get the location of the folder that contains the mlreportgen.report.HTMLModuleTabs reporter class definition.

```
path = mlreportgen.report.HTMLModuleTabs.getClassFolder();
```

### **See Also**

mlreportgen.report.HTMLModuleTabs | mlreportgen.report.Report

**Introduced in R2020a**



# mlreportgen.report.ListOfCaptions.createTemplate

**Class:** mlreportgen.report.ListOfCaptions

**Package:** mlreportgen.report

Create list of captions reporter template

## Syntax

```
template = mlreportgen.report.ListOfCaptions.createTemplate(templatePath,
type)
```

## Description

`template = mlreportgen.report.ListOfCaptions.createTemplate(templatePath, type)` creates a copy of the `mlreportgen.report.ListOfCaptions` reporter template for the report type specified by `type` at the location specified by `templatePath`. You can use the copied template as a starting point to design a custom list of captions reporter template for your report.

## Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

"html" | "html-file" | "docx" | "pdf"

Type of template, specified as "html", "html-file", "docx", or "pdf".

## Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

## Examples

### Create a List of Captions Reporter Template

Create a copy of the HTML template for the `mlreportgen.report.ListOfCaptions` reporter and save it with the name `myListOfCaptionsTemplate` in the `mytemplates` folder.

```
template = mlreportgen.report.ListOfCaptions.createTemplate...
('mytemplates/myListOfCaptionsTemplate', 'html');
```

After you modify the template, you can use it by setting the `TemplateSrc` property of a `ListOfCaptions` reporter to the path of the template file.

### **See Also**

`mlreportgen.report.ListOfCaptions`

**Introduced in R2020b**

# mlreportgen.report.ListOfCaptions.customizeReporter

**Class:** mlreportgen.report.ListOfCaptions

**Package:** mlreportgen.report

Create custom list of captions reporter class

## Syntax

```
reporter = mlreportgen.report.ListOfCaptions.customizeReporter(classpath)
```

## Description

`reporter = mlreportgen.report.ListOfCaptions.customizeReporter(classpath)` creates a class definition file that defines a subclass of `mlreportgen.report.ListOfCaptions` at the location specified by `classpath`. This method also copies the default reporter templates to the `resources/templates` subfolder of the folder that contains the class definition file. You can use the class definition file as a starting point to design a custom list of captions reporter class for your report.

## Input Arguments

**classpath — Path and name of new class definition file**

string scalar | character vector

Path and name of new class definition file, specified as a string scalar or character vector.

You can specify a relative path or an absolute path. For example, this code creates `MyClass.m` in the subfolder `myFolder` of the current folder.

```
mlreportgen.report.ListOfCaptions.customizeReporter("myFolder/MyClass")
```

To create the reporter class in a class folder, precede the class name with the `@` character. Do not specify the `.m` extension. For example, this code creates `MyClass.m` in the subfolder `myFolder/@MyClass` in the current folder.

```
mlreportgen.report.ListOfCaptions.customizeReporter("myFolder/@MyClass")
```

See “Folders Containing Class Definitions”.

To create the reporter class in a class package, precede the folder name with the `+` character. For example, this code creates an list of captions reporter in the `myOrg` package folder in the current folder.

```
mlreportgen.report.ListOfCaptions.customizeReporter("+myOrg/@MyClass");
```

## Output Arguments

**reporter — Path and file name of new list of captions reporter class**

string scalar

Path and file name of the new list of captions reporter class, returned as a string scalar.

## Examples

### Create Custom List of Captions Reporter

Create a custom list of captions reporter, `MyListOfCaptions`, and the associated default templates in the subfolder `MyFolder` of the current working folder.

```
mlreportgen.report.ListOfCaptions.customizeReporter('MyFolder/MyListOfCaptions')
```

```
ans =
```

```
    "MyFolder\MyListOfCaptions.m"
```

### See Also

`mlreportgen.report.ListOfCaptions`

**Introduced in R2020b**

# mlreportgen.report.ListOfCaptions.getClassFolder

**Class:** mlreportgen.report.ListOfCaptions

**Package:** mlreportgen.report

Get location of list of captions reporter class definition file

## Syntax

```
path = mlreportgen.report.ListOfCaptions.getClassFolder()
```

## Description

`path = mlreportgen.report.ListOfCaptions.getClassFolder()` returns the path of the folder that contains the `mlreportgen.report.ListOfCaptions` class definition file.

## Output Arguments

**path** — Location of the list of captions reporter class definition file

character vector

Location of the `mlreportgen.report.ListOfCaptions` class definition file, returned as a character vector.

## Examples

### Get List of Captions Reporter Class Folder

Get the folder that contains the `mlreportgen.report.ListOfCaptions` reporter class definition.

```
path = mlreportgen.report.ListOfCaptions.getClassFolder();
```

## See Also

`mlreportgen.report.ListOfCaptions`

**Introduced in R2020b**

## getTitleReporter

**Class:** `mlreportgen.report.ListOfCaptions`

**Package:** `mlreportgen.report`

Get list of captions title reporter

### Syntax

```
titleReporter = getTitleReporter(loc)
```

### Description

`titleReporter = getTitleReporter(loc)` returns the `mlreportgen.report.Title` reporter that the specified `mlreportgen.report.ListOfCaptions` reporter uses to format the list of captions section title. Use the `Title` reporter to customize the alignment, position, and appearance of the title.

### Input Arguments

**loc** — List of captions reporter

`mlreportgen.report.ListOfCaptions` object

List of captions reporter, specified as an `mlreportgen.report.ListOfCaptions` object.

### Output Arguments

**titleReporter** — List of captions title reporter

`mlreportgen.report.Title` object

List of captions title reporter, returned as an `mlreportgen.report.Title` object.

### Examples

#### Use a Custom Template for a List of Captions Title

Suppose that you defined a custom template for a list of captions title in the template library of a template for your report output type. To use the custom template:

- 1 Get the title reporter using `getTitleReporter`.
- 2 Set the properties of the title reporter. Set the `Content` property to the title text, the `TemplateSrc` property to the path and file name of the template file, and the `TemplateName` property to the name of the custom template for the list of captions title.
- 3 Set the `Title` property of the list of captions reporter to the title reporter.

For example, suppose that the template file `myTemplate.pdf` defines two templates to use with a list of captions title. One template, `ListOfCaptionsTitleRed`, uses a style that makes the title red.

The second template, `ListOfCaptionsTitleBlue`, uses a style that makes the title blue. This code applies the template that makes the title blue:

```
import mlreportgen.report.*
import mlreportgen.dom.*
rpt = Report('myReport', 'pdf');
open(rpt);

loc = ListOfCaptions;
loc.AutoNumberStreamName = "equation";

titleReporter = getTitleReporter(loc);
titleReporter.Content = "List of Equations";
titleReporter.TemplateSrc = "myTemplate.pdf";
titleReporter.TemplateName = "ListOfCaptionsTitleBlue";
loc.Title = titleReporter;
append(rpt, loc);

ch = Chapter("Physics Principles")
eq = Equation("e = m * c^2");
eq.DisplayInline = true;
append(ch, eq);

p = Paragraph("Equation ");
p.Style = {CounterInc('equation'), WhiteSpace("preserve")};
append(p, AutoNumber("equation"));
append(p, " Mass-energy equivalence");
append(ch, p);
append(rpt, ch);

close(rpt);
rptview(rpt);
```

## See Also

`mlreportgen.report.ListOfCaptions`

## Topics

“Create an HTML Document Part Template Library” on page 13-30

“Create a PDF Document Part Template Library” on page 13-32

“Create a Microsoft Word Document Part Template Library” on page 13-27

## Introduced in R2020b

## mlreportgen.report.ListOfFigures.createTemplate

**Class:** mlreportgen.report.ListOfFigures

**Package:** mlreportgen.report

Create list of figures reporter template

### Syntax

```
template = mlreportgen.report.ListOfFigures.createTemplate(templatePath,type)
```

### Description

`template = mlreportgen.report.ListOfFigures.createTemplate(templatePath,type)` creates a copy of the `mlreportgen.report.ListOfFigures` reporter template for the report type specified by `type` at the location specified by `templatePath`. You can use the copied template as a starting point to design a custom list of figures reporter template for your report.

### Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

"html" | "html-file" | "docx" | "pdf"

Type of template, specified as "html", "html-file", "docx", or "pdf".

### Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

### Examples

#### Create a List of Figures Reporter Template

Create a copy of the HTML template for the `mlreportgen.report.ListOfFigures` reporter and save it with the name `myListOfFiguresTemplate` in the `mytemplates` folder.

```
template = mlreportgen.report.ListOfFigures.createTemplate...  
('mytemplates/myListOfFiguresTemplate','html');
```



After you modify the template, you can use it by setting the `TemplateSrc` property of a `ListOfFigures` reporter to the path of the template file.

### **See Also**

`mlreportgen.report.ListOfFigures`

**Introduced in R2020b**

## mlreportgen.report.ListOfFigures.customizeReporter

**Class:** mlreportgen.report.ListOfFigures

**Package:** mlreportgen.report

Create custom list of figures reporter class

### Syntax

```
reporter = mlreportgen.report.ListOfFigures.customizeReporter(classpath)
```

### Description

`reporter = mlreportgen.report.ListOfFigures.customizeReporter(classpath)` creates a class definition file that defines a subclass of `mlreportgen.report.ListOfFigures` at the location specified by `classpath`. This method also copies the default reporter templates to the `resources/templates` subfolder of the folder that contains the class definition file. You can use the class definition file as a starting point to design a custom list of figures reporter class for your report.

### Input Arguments

**classpath — Path and name of new class definition file**

string scalar | character vector

Path and name of new class definition file, specified as a string scalar or character vector.

You can specify a relative path or an absolute path. For example, this code creates `MyClass.m` in the subfolder `myFolder` of the current folder.

```
mlreportgen.report.ListOfFigures.customizeReporter("myFolder/MyClass");
```

To create the reporter class in a class folder, precede the class name with the `@` character. Do not specify the `.m` extension. For example, this code creates `MyClass.m` in the subfolder `myFolder/@MyClass` in the current folder.

```
mlreportgen.report.ListOfFigures.customizeReporter("myFolder/@MyClass");
```

See “Folders Containing Class Definitions”.

To create the reporter class in a class package, precede the folder name with the `+` character. For example, this code creates an list of figures reporter in the `myOrg` package folder in the current folder.

```
mlreportgen.report.ListOfFigures.customizeReporter("+myOrg/@MyClass");
```

### Output Arguments

**reporter — Path and file name of new list of figures reporter class**

string scalar

Path and file name of the new list of figures reporter class, returned as a string scalar.

## Examples

### Create Custom List of Figures Reporter

Create a custom list of figures reporter, `MyListOfFigures`, and its associated default templates in the subfolder `MyFolder` of the current working folder.

```
mlreportgen.report.ListOfFigures.customizeReporter('MyFolder/MyListOfFigures')
```

```
ans =
```

```
    "MyFolder\MyListOfFigures.m"
```

### See Also

`mlreportgen.report.ListOfFigures`

**Introduced in R2020b**

## **mlreportgen.report.ListOfFigures.getClassFolder**

**Class:** mlreportgen.report.ListOfFigures

**Package:** mlreportgen.report

Get location of list of figures reporter class definition file

### **Syntax**

```
path = mlreportgen.report.ListOfFigures.getClassFolder()
```

### **Description**

`path = mlreportgen.report.ListOfFigures.getClassFolder()` returns the path of the folder that contains the `mlreportgen.report.ListOfFigures` class definition file.

### **Output Arguments**

**path** — Location of the list of figures reporter class definition file

character vector

Location of the `mlreportgen.report.ListOfFigures` class definition file, returned as a character vector.

### **Examples**

#### **Get List of Figures Reporter Class Folder**

Get the folder that contains the `mlreportgen.report.ListOfFigures` reporter class definition.

```
path = mlreportgen.report.ListOfFigures.getClassFolder();
```

### **See Also**

`mlreportgen.report.ListOfFigures`

**Introduced in R2020b**

# getTitleReporter

**Class:** `mlreportgen.report.ListOfFigures`

**Package:** `mlreportgen.report`

Get list of figures title reporter

## Syntax

```
titleReporter = getTitleReporter(lof)
```

## Description

`titleReporter = getTitleReporter(lof)` returns the `mlreportgen.report.Title` reporter that the specified `mlreportgen.report.ListOfFigures` reporter uses to format the list of figures section title. Use the returned `Title` reporter to customize the alignment, position, and appearance of the title.

## Input Arguments

**lof** — List of figures reporter

`mlreportgen.report.ListOfFigures` object

List of figures reporter, specified as an `mlreportgen.report.ListOfFigures` object.

## Output Arguments

**titleReporter** — List of figures title reporter

`mlreportgen.report.Title` object

List of figures reporter, returned as an `mlreportgen.report.Title` object.

## Examples

### Use a Custom Template for a List of Figures Title

Suppose that you defined a custom template for a list of figures title in the template library of a template for your report output type. To use the custom template:

- 1 Get the title reporter using `getTitleReporter`.
- 2 Set the properties of the title reporter. Set the `Content` property to the title text, the `TemplateSrc` property to the path and file name of the template file, and the `TemplateName` property to the name of the custom template for the list of figures title.
- 3 Set the `Title` property of the list of figures reporter to the title reporter.

For example, suppose that the template file `myTemplate.pdf` defines two templates to use with a list of figures title. One template, `ListOfFiguresTitleRed`, uses a style that makes the title red.

The second template, `ListOfFiguresTitleBlue`, makes the title blue. This code applies the template that makes the title blue:

```
import mlreportgen.report.*
rpt = Report('myReport', 'pdf');
open(rpt);

lof = ListOfFigures();
titleReporter = getTitleReporter(lof);
titleReporter.Content = "My List of Figures";
titleReporter.TemplateSrc = "myTemplate.pdfctx";
titleReporter.TemplateName = "ListOfFiguresTitleBlue";
lof.Title = titleReporter;
append(rpt, lof);

ch = Chapter("Vegetables");
f = FormalImage(which("peppers.png"));
f.Caption = "Peppers";
append(ch, f);
append(rpt, ch);
close(rpt);
rptview(rpt);
```

## See Also

`mlreportgen.report.ListOfFigures`

## Topics

“Create an HTML Document Part Template Library” on page 13-30

“Create a PDF Document Part Template Library” on page 13-32

“Create a Microsoft Word Document Part Template Library” on page 13-27

## Introduced in R2020b

# mlreportgen.report.ListOfTables.createTemplate

**Class:** mlreportgen.report.ListOfTables

**Package:** mlreportgen.report

Create list of tables reporter template

## Syntax

```
template = mlreportgen.report.ListOfTables.createTemplate(templatePath,type)
```

## Description

`template = mlreportgen.report.ListOfTables.createTemplate(templatePath,type)` creates a copy of the `mlreportgen.report.ListOfTables` reporter template for the report type specified by `type` at the location specified by `templatePath`. You can use the copied template as a starting point to design a custom list of tables reporter template for your report.

## Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

"html" | "html-file" | "docx" | "pdf"

Type of template, specified as "html", "html-file", "docx", or "pdf".

## Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

## Examples

### Create a List of Tables Reporter Template

Create a copy of the HTML template for the `mlreportgen.report.ListOfTables` reporter and save it with the name `myListOfTablesTemplate` in the `mytemplates` folder.

```
template = mlreportgen.report.ListOfTables.createTemplate...  
('mytemplates/myListOfTablesTemplate','html');
```

After you modify the template, you can use it by setting the `TemplateSrc` property of a `ListOfTables` reporter to the path of the template file.

### **See Also**

`mlreportgen.report.ListOfTables`

**Introduced in R2020b**



# mlreportgen.report.ListOfTables.customizeReporter

**Class:** mlreportgen.report.ListOfTables

**Package:** mlreportgen.report

Create custom list of tables reporter class

## Syntax

```
reporter = mlreportgen.report.ListOfTables.customizeReporter(classpath)
```

## Description

`reporter = mlreportgen.report.ListOfTables.customizeReporter(classpath)` creates a class definition file that defines a subclass of `mlreportgen.report.ListOfTables` at the location specified by `classpath`. This method also copies the default reporter templates to the `resources/templates` subfolder of the folder that contains the class definition file. You can use the class definition file as a starting point to design a custom list of tables reporter class for your report.

## Input Arguments

**classpath — Path and name of new class definition file**

string scalar | character vector

Path and name of new class definition file, specified as a string scalar or character vector.

You can specify a relative path or an absolute path. For example, this code creates `MyClass.m` in the subfolder `myFolder` of the current folder.

```
mlreportgen.report.ListOfTables.customizeReporter("myFolder/MyClass");
```

To create the reporter class in a class folder, precede the class name with the `@` character. Do not specify the `.m` extension. For example, this code creates `MyClass.m` in the subfolder `myFolder/@MyClass` in the current folder.

```
mlreportgen.report.ListOfTables.customizeReporter("myFolder/@MyClass");
```

See “Folders Containing Class Definitions”.

To create the reporter class in a class package, precede the folder name with the `+` character. For example, this code creates a list of tables reporter in the `myOrg` package folder in the current folder.

```
mlreportgen.report.ListOfTables.customizeReporter("+myOrg/@MyClass");
```

## Output Arguments

**reporter — Path and file name of new list of tables reporter class**

string scalar

Path and file name of the new list of tables reporter class, returned as a string scalar.

## Examples

### Create Custom List of Tables Reporter

Create a custom list of tables reporter, `MyListOfTables`, and the associated default templates in the subfolder `MyFolder` of the current working folder.

```
mlreportgen.report.ListOfTables.customizeReporter('MyFolder/MyListOfTables')
```

```
ans =
```

```
    "MyFolder\MyListOfTables.m"
```

### See Also

`mlreportgen.report.ListOfTables`

**Introduced in R2020b**

# mlreportgen.report.ListOfTables.getClassFolder

**Class:** mlreportgen.report.ListOfTables

**Package:** mlreportgen.report

Get location of list of tables reporter class definition file

## Syntax

```
path = mlreportgen.report.ListOfTables.getClassFolder()
```

## Description

`path = mlreportgen.report.ListOfTables.getClassFolder()` returns the path of the folder that contains the `mlreportgen.report.ListOfTables` class definition file.

## Output Arguments

**path** — Location of the list of tables reporter class definition file

character vector

Location of the `mlreportgen.report.ListOfTables` class definition file, returned as a character vector.

## Examples

### Get List of Tables Reporter Class Folder

Get the folder that contains the `mlreportgen.report.ListOfTables` reporter class definition.

```
path = mlreportgen.report.ListOfTables.getClassFolder();
```

## See Also

`mlreportgen.report.ListOfTables`

**Introduced in R2020b**

## getTitleReporter

**Class:** `mlreportgen.report.ListOfTables`

**Package:** `mlreportgen.report`

Get list of tables title reporter

### Syntax

```
titleReporter = getTitleReporter(lot)
```

### Description

`titleReporter = getTitleReporter(lot)` returns the `mlreportgen.report.Title` reporter that the specified `mlreportgen.report.ListOfTables` reporter uses to format the list of tables title. Use the properties of the returned `Title` reporter to customize the alignment, position, and appearance of the title.

### Input Arguments

**tot** — List of tables reporter

`mlreportgen.report.ListOfTables` object

List of tables reporter, specified as an `mlreportgen.report.ListOfTables` object.

### Output Arguments

**titleReporter** — List of tables title reporter

`mlreportgen.report.Title` object

List of tables reporter, returned as an `mlreportgen.report.Title` object.

## Examples

### Use a Custom Template for a List of Tables Title

Suppose that you defined a custom template for a list of tables title in the template library of a template for your report output type. To use the custom template:

- 1 Get the title reporter using `getTitleReporter`.
- 2 Set the properties of the title reporter. Set the `Content` property to the title text, the `TemplateSrc` property to the path and file name of the template file, and the `TemplateName` property to the name of the custom template for the list of tables title.
- 3 Set the `Title` property of the list of tables reporter to the title reporter.

For this example, suppose that the template file `myTemplate.pdf` defines two templates to use with a list of tables title. One template, `ListOfTablesTitleRed`, uses a style that makes the title

red. The second template, `ListOfTablesTitleBlue`, uses a style that makes the title blue. This code applies the template that makes the title blue:

```
import mlreportgen.report.*
rpt = Report('myReport', 'pdf');
open(rpt);

lot = ListOfTables();
titleReporter = getTitleReporter(lot);
titleReporter.Content = "My List of Tables Title";
titleReporter.TemplateSrc = "myTemplate.pdf";
titleReporter.TemplateName = "ListOfTablesTitleBlue";
lot.Title = titleReporter;
append(rpt,lot);

ch = Chapter("Magic Squares");
t = BaseTable(magic(2));
t.Title = "Order Two Magic Square";
append(ch,t);

append(rpt,ch);
close(rpt);
rptview(rpt);
```

## See Also

`mlreportgen.report.ListOfTables`

## Topics

“Create an HTML Document Part Template Library” on page 13-30

“Create a PDF Document Part Template Library” on page 13-32

“Create a Microsoft Word Document Part Template Library” on page 13-27

## Introduced in R2020b

## mlreportgen.report.MATLABCode.createTemplate

**Class:** mlreportgen.report.MATLABCode

**Package:** mlreportgen.report

Create copy of mlreportgen.report.MATLABCode reporter template

### Syntax

```
template = mlreportgen.report.MATLABCode.createTemplate(templatePath,type)
```

### Description

`template = mlreportgen.report.MATLABCode.createTemplate(templatePath,type)` creates a copy of the mlreportgen.report.MATLABCode reporter template for the report type specified by `type` at the location specified by `templatePath`. You can use the copied template as a starting point to design a custom MATLABCode reporter template for your report.

### Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

"html" | "html-file" | "docx" | "pdf"

Type of template, specified as "html", "html-file", "docx", or "pdf".

### Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

### Examples

#### Create MATLAB Code Reporter Template

Create a copy of the HTML template for the mlreportgen.report.MATLABCode reporter and save it with the name myMATLABCodeTemplate in the mytemplates folder.

```
template = mlreportgen.report.MATLABCode.createTemplate...  
('mytemplates/myMATLABCodeTemplate','html');
```

After you modify the template, you can use it by setting the `TemplateSrc` property of a `MATLABCode` reporter to the path of the template file.

**See Also**

`mlreportgen.report.MATLABCode`

**Introduced in R2021a**

# mlreportgen.report.MATLABCode.customizeReporter

**Class:** mlreportgen.report.MATLABCode

**Package:** mlreportgen.report

Create subclass of mlreportgen.report.MATLABCode class

## Syntax

```
reporter = mlreportgen.report.MATLABCode.customizeReporter(classpath)
```

## Description

`reporter = mlreportgen.report.MATLABCode.customizeReporter(classpath)` creates a reporter class definition file that defines a subclass of `mlreportgen.report.MATLABCode` at the location specified by `classpath`. The method also copies the default reporter templates to the `resources/templates` subfolder of the folder that contains the class definition file. You can use the class definition file as a starting point to design a custom MATLAB code reporter class for your report.

## Input Arguments

**classpath — Path and name of new class definition file**

string scalar | character vector

Path and name of the new class definition file, specified as a string scalar or character vector.

You can specify a relative path or an absolute path. For example, this code creates `MyClass.m` in the subfolder `myFolder` of the current folder.

```
mlreportgen.report.MATLABCode.customizeReporter("myFolder/MyClass")
```

To create the reporter class in a class folder, precede the class name with the `@` character. Do not specify the `.m` extension. For example, this code creates `MyClass.m` in the subfolder `myFolder/@MyClass` in the current folder.

```
mlreportgen.report.MATLABCode.customizeReporter("myFolder/@MyClass")
```

See “Folders Containing Class Definitions”.

To create the reporter class in a class package, precede the folder name with the `+` character. For example, this code creates a MATLAB code reporter in the `myOrg` package folder in the current folder.

```
mlreportgen.report.MATLABCode.customizeReporter("+myOrg/@MyClass")
```

## Output Arguments

**reporter — Path and file name of new reporter class**

string scalar



Path and file name of the new reporter class, returned as a string scalar.

## Examples

### Create Custom MATLAB Code Reporter

Create a custom `mlreportgen.report.MATLABCode` reporter, `myMATLABCode`, and the associated default templates in the subfolder `MyFolder` of the current working folder.

```
mlreportgen.report.MATLABCode.customizeReporter("myFolder/myMATLABCode")
```

```
ans =
```

```
    "myFolder\myMATLABCode.m"
```

### See Also

`mlreportgen.report.MATLABCode`

**Introduced in R2021a**

## **mlreportgen.report.MATLABCode.getClassFolder**

**Class:** mlreportgen.report.MATLABCode

**Package:** mlreportgen.report

Get location of folder that contains mlreportgen.report.MATLABCode class definition file

### **Syntax**

```
path = mlreportgen.report.MATLABCode.getClassFolder()
```

### **Description**

path = mlreportgen.report.MATLABCode.getClassFolder() returns the path of the folder that contains the mlreportgen.report.MATLABCode class definition file.

### **Output Arguments**

**path** — mlreportgen.report.MATLABCode class definition file location

character vector

mlreportgen.report.MATLABCode class definition file location, returned as a character vector.

### **Examples**

#### **Get mlreportgen.report.MATLABCode Class Folder**

Get the location of the folder that contains the mlreportgen.report.MATLABCode reporter class definition file.

```
path = mlreportgen.report.MATLABCode.getClassFolder()
```

### **See Also**

mlreportgen.report.MATLABCode

**Introduced in R2021a**

# mlreportgen.report.MATLABVariable.createTemplate

**Class:** mlreportgen.report.MATLABVariable

**Package:** mlreportgen.report

Create MATLAB variable template

## Syntax

```
template = mlreportgen.report.MATLABVariable.createTemplate(templatePath,  
type)
```

## Description

`template = mlreportgen.report.MATLABVariable.createTemplate(templatePath, type)` creates a copy of the MATLAB variable reporter template specified by `type` at the `templatePath` location. You can use the copied template as a starting point to design a custom MATLAB variable template for your report.

## Input Arguments

**templatePath** — Path and file name of new template

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type** — Type of template

"html" | "html-file" | "docx" | "pdf"

Type of template, specified as "html", "html-file", "docx", or "pdf".

## Output Arguments

**template** — Path and file name of template copy

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdf.tx`.

## Examples

### Create MATLAB Variable Template

To use the new template for the MATLAB variable, assign its path to the `TemplateSrc` property of `mlreportgen.report.MATLABVariable`.

```
import mlreportgen.report.*  
rpt = Report('My Report', 'html');
```

```
mlvar = MATLABVariable();  
template = MATLABVariable.createTemplate...  
    ('mytemplates\myMLVar','html');  
mlvar.TemplateSrc = template;
```

## See Also

`mlreportgen.report.Reporter` | `mlreportgen.report.Report` |  
`mlreportgen.report.MATLABVariable`

## Topics

“Modify Styles in a Microsoft Word Template” on page 13-126

“Modify Styles in HTML Templates” on page 13-137

“Modify Styles in PDF Templates” on page 13-138

## Introduced in R2018b

# mlreportgen.report.MATLABVariable.customizeReporter

**Class:** mlreportgen.report.MATLABVariable

**Package:** mlreportgen.report

Create custom MATLAB variable reporter class

## Syntax

```
rptr = mlreportgen.report.MATLABVariable.customizeReporter(classpath)
```

## Description

`rptr = mlreportgen.report.MATLABVariable.customizeReporter(classpath)` creates a MATLAB variable class definition file that is a subclass of `mlreportgen.report.MATLABVariable`. The file is created at the specified `classpath` location. The `MATLABVariable.customizeReporter` method also copies the default `MATLABVariable` templates to the `<classpath>/resources/template` folder. You can use the class definition file as a starting point to design a custom `MATLABVariable` class for your report.

## Input Arguments

**classpath** — Location of custom MATLAB variable class

current working folder (default) | string | character array

Location of custom MATLAB variable class, specified as a string or character array. The `classpath` argument also supports specifying a folder with @ before the class name.

## Output Arguments

**rptr** — MATLAB variable reporter path

string

MATLAB variable reporter path, returned as the string specifying the path to the derived report class file.

## Examples

### Create Custom MATLAB Variable Reporter

Create a custom MATLAB variable reporter and its associated default templates. The derived class file is created at the specified path relative to the current working folder. In this case, the path to the `MyMATLABvar.m` class file is `<current working folder>/newVar/@MyMATLABvar/MyMATLABvar.m`. The default title page templates are in the `<current working folder>/newVar/@MyMATLABvar/resources/templates` folder.

```
import mlreportgen.report.*
MATLABVariable.customizeReporter('newVar/@MyMATLABvar')
```

After editing this new class file, you can use it as your MATLAB variable reporter.

```
mvar = MyMATLABvar();
```

**See Also**

`mlreportgen.report.MATLABVariable` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Report`

**Introduced in R2018b**

# mlreportgen.report.MATLABVariable.getClassFolder

**Class:** mlreportgen.report.MATLABVariable

**Package:** mlreportgen.report

MATLAB variable class definition file location

## Syntax

```
path = mlreportgen.report.MATLABVariable.getClassFolder()
```

## Description

`path = mlreportgen.report.MATLABVariable.getClassFolder()` returns the path of the folder that contains the MATLAB variable class definition file.

## Output Arguments

**path** — MATLAB variable class definition file location

character array

MATLAB variable class definition file location, returned as a character array.

## See Also

mlreportgen.report.MATLABVariable | mlreportgen.report.Reporter | mlreportgen.report.Report

**Introduced in R2018b**

## getVariableValue

**Class:** mlreportgen.report.MATLABVariable

**Package:** mlreportgen.report

Get MATLAB variable value

### Syntax

```
value = getVariableValue(rptr)
```

### Description

`value = getVariableValue(rptr)` returns the value variable specified by the `Variable` and `Location` properties of the specified `MATLABVariable` reporter (`rptr`).

### Input Arguments

**rptr — MATLAB variable reporter name**

mlreportgen.report.MATLABVariable reporter

mlreportgen.report.MATLABVariable reporter name.

### Output Arguments

**value — MATLAB variable value**

depends on variable

MATLAB variable value. The data type of the returned value depends on the data type of the variable.

### Examples

#### Obtain the Value of a MATLAB Workspace Variable

```
x = 17;  
rptr = mlreportgen.report.MATLABVariable(x);  
value = getVariableValue(rptr)
```

```
value =
```

```
    17
```

### See Also

mlreportgen.report.MATLABVariable | mlreportgen.report.Reporter |  
mlreportgen.report.Report

**Introduced in R2018b**



# setVariableValue

**Class:** `mlreportgen.report.MATLABVariable`

**Package:** `mlreportgen.report`

Set the value to report for a variable

## Syntax

```
setVariableValue(rptr,value)
```

## Description

`setVariableValue(rptr,value)` sets the value of the variable reported by the specified reporter to the specified value and sets the `Location` property of the reporter to "User-Defined".

## Input Arguments

**rptr** — MATLAB variable reporter

`mlreportgen.report.MATLABVariable` object

MATLAB variable reporter, specified as an `mlreportgen.report.MATLABVariable` object.

**value** — Value to report

(default) | Any MATLAB data type

Value to report, specified as any MATLAB data type.

## Examples

### Set Value for MATLABVariable Reporter to Report

This example shows how to report values that are not assigned to variables in the MATLAB® workspace. The example creates a `containers.Map` object. The `Map` object is defined in the MATLAB workspace, but the key-value pair entries are not assigned to variables in the workspace. To report on the entries, the example creates a reporter for each entry in the `Map` object, gets the entry value from the `Map` object, and then uses `setVariableValue` to specify the value to report.

Import the Report API package so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
```

Create a `Map` object.

```
myMap = containers.Map(["key1", "key2"], [100, 200]);
```

Create a report.

```
report = mlreportgen.report.Report("My Report", "PDF");
```

Determine the number of key-value pair entries in the `Map` object.

```
n = length(myMap);
```

Return the keys of the Map object in a cell array.

```
myKeys = keys(myMap);
```

Create a reporter for each entry in the Map object. Use `setVariableValue` to set the value to report for the entry.

```
for i=1:n
    myKey = myKeys{i};
    rptr = mlreportgen.report.MATLABVariable();
    rptr.Variable = myKey;
    setVariableValue(rptr,myMap(myKey));
    append(report,rptr);
end
```

Close and view the report.

```
close(report);
rptview(report);
```

Here are the key-value pairs in the report:

```
key1. 100
```

```
key2. 200
```

## See Also

`mlreportgen.report.MATLABVariable`

**Introduced in R2020b**

# add

**Class:** `mlreportgen.report.Report`

**Package:** `mlreportgen.report`

(Not recommended) Add content to report

## Syntax

```
add(report, content)
```

## Description

---

**Note** `add` is not recommended. Use `append` instead. See “Compatibility Considerations” on page 11-268.

---

`add(report, content)` adds the specified content to the specified report. If the report is not already open, the `add` method opens it.

---

**Note** You can add a reporter to a report multiple times, but you cannot add the reporter to different reports. For example, if you add an `mlreportgen.report.TitlePage` reporter to one report, you cannot add it to another report.

---

## Input Arguments

### **report** — Report

`mlreportgen.report.Report` object

Report, specified as an `mlreportgen.report.Report` object.

### **content** — Content to add to report

Report API reporter | DOM object | MATLAB built-in object

Content to add to report, specified as a Report API reporter, DOM object, or built-in MATLAB object. The content can be a Report API reporter or any object that you can append to a DOM document. Content that you can append to a DOM document includes DOM objects and many built-in MATLAB objects, such as strings, character arrays, and cell arrays.

## Examples

### **Add Content to a Report**

Add content to a paragraph and add the paragraph to a report. To add the content to the paragraph, you must use `append` because a paragraph is a DOM API object. To add the paragraph to the report, which is a Report API object, this example uses `add`. Starting in R2020b, you can use `append` instead of `add`. See “Compatibility Considerations” on page 11-268.

```

import mlreportgen.dom.*
import mlreportgen.report.*

rpt = Report("My Report");
p = Paragraph("My paragraph content ");

append(p,"and some more content.");
add(rpt,p);

close(rpt);
rptview(rpt);

```

## Compatibility Considerations

### add method is not recommended

*Not recommended starting in R2020b*

Starting in R2020b, use the `append` method instead of the `add` method to add content to objects of these Report API classes:

- `mlreportgen.report.Report`
- `mlreportgen.report.Chapter`
- `mlreportgen.report.Section`

To add content to a DOM API object, such as an `mlreportgen.dom.Paragraph` object, continue to use the `append` method of the DOM object. The advantage of using `append` to add content to Report API objects is that you use the same method name as you use to add content to DOM API objects.

There are no plans to remove the `add` methods of the `Report`, `Chapter`, or `Section` classes. Report API programs that use the `add` methods will continue to run.

To update existing code, replace the method name `add` with `append` as shown by the examples in the table.

Not Recommended	Recommended
<pre> import mlreportgen.report.* import mlreportgen.dom.*  rpt = Report("My Report","pdf"); ch = Chapter("My Chapter"); sect = Section("My Section"); para = Paragraph("My Content "); add(para,"more Content"); add(sect,para); add(ch,sect); add(rpt,ch);  close(rpt); rptview(rpt); </pre>	<pre> import mlreportgen.report.* import mlreportgen.dom.*  rpt = Report("My Report","pdf"); ch = Chapter("My Chapter"); sect = Section("My Section"); para = Paragraph("My Content "); append(para,"more Content"); append(sect,para); append(ch,sect); append(rpt,ch);  close(rpt); rptview(rpt); </pre>

## See Also

`mlreportgen.report.Report` | `append` | `open` | `close` | `rptview`

**Topics**

"Add Content to a Report" on page 13-10

**Introduced in R2017b**

## append

**Class:** `mlreportgen.report.Report`

**Package:** `mlreportgen.report`

Add content to report

### Syntax

```
append(report, content)
```

### Description

`append(report, content)` adds the specified content to the specified report. If the report is not already open, the `append` method opens it.

---

**Note** You can add a reporter to a report multiple times, but you cannot add the reporter to different reports. For example, if you add an `mlreportgen.report.TitlePage` reporter to one report, you cannot add it to another report.

---

### Input Arguments

#### **report** — Report

`mlreportgen.report.Report` object

Report, specified as an `mlreportgen.report.Report` object.

#### **content** — Content to add to report

Report API reporter | DOM object | MATLAB built-in object

Content to add to report, specified as a Report API reporter, DOM object, or built-in MATLAB object. The content can be a Report API reporter or any object that you can append to a DOM document. Content that you can append to a DOM document includes DOM objects and many built-in MATLAB objects, such as strings, character arrays, and cell arrays.

### Examples

#### **Add Content to a Report**

This example adds a title page, table of contents, and chapter to a report by using the `append` method of the `mlreportgen.report.Report` object. To add content to the chapter, the example uses the `append` method of the `mlreportgen.report.Chapter` object.

Import the DOM and API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*  
import mlreportgen.dom.*;
```

Create a report and add a title and table of contents to the report.

```
rpt = Report("Magic Square Magic");
append(rpt, TitlePage("Title", "Magic Square Magic", "Subtitle", ...
    "Inverting a Magic Square", "Author", "John Doe"));
append(rpt, TableOfContents);
```

Create a chapter and add content to the chapter.

```
ch = Chapter("Magic Moments");
m = magic(5);
append(ch, BaseTable("Title", "m = magic(5)", "Content", m));
mInverse = m^-1;
append(ch, BaseTable("Title", "mInverse = magic(5)^-1", "Content", ...
    cellfun(@(x) sprintf("%0.3f", x), num2cell(mInverse), ...
    "UniformOutput", false)));
append(ch, BaseTable("Title", "m*mInverse", "Content", ...
    cellfun(@(x) sprintf("%0.3f", x), num2cell(m*mInverse), ...
    "UniformOutput", false)));
append(ch, Paragraph(sprintf("sum(m(1,:)) - sum(m(:,1)) = %d", ...
    sum(m(1,:)) - sum(m(:,1)))));
append(ch, Paragraph(sprintf("sum(mInverse(1,:)) - sum(mInverse(:,1)) = %0.3f", ...
    sum(mInverse(1,:)) - sum(mInverse(:,1)))));
```

Add the chapter to the report.

```
append(rpt, ch);
```

Close and view the report.

```
close(rpt);
rptview(rpt);
```

## See Also

`mlreportgen.report.Report` | `open` | `close` | `rptview`

## Topics

“Add Content to a Report” on page 13-10

## Introduced in R2020b

## close

**Class:** `mlreportgen.report.Report`

**Package:** `mlreportgen.report`

Close and generate report

### Syntax

```
close(report)
```

### Description

`close(report)` closes the report and generates its content. The generated report is a file of the type specified by the `Type` property of the report. The report is at the location specified by the `OutputPath` property of the report. For information on these properties, see `mlreportgen.report.Report`.

### Input Arguments

**report — Report**

`report` object

Report, specified as a report object. To create the report input, use `mlreportgen.report.Report`.

### See Also

`mlreportgen.report.Report` | `append` | `open` | `fill`

**Introduced in R2017b**



# mlreportgen.report.Report.createTemplate

**Class:** mlreportgen.report.Report

**Package:** mlreportgen.report

Create report template

## Syntax

```
template = mlreportgen.report.Report.createTemplate(templatePath,type)
```

## Description

`template = mlreportgen.report.Report.createTemplate(templatePath,type)` creates a copy of the default report template specified by `type` at the location specified by `templatePath`. You can use the copied template to design a custom template for your report.

## Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

"html" | "html-file" | "docx" | "pdf"

Type of template, specified as "html", "html-file", "docx", or "pdf".

## Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdfx`.

## Examples

### Create a Report Template

Create a copy of the HTML template for a report and save it with the name `myReportTemplate` in the `mytemplates` folder.

```
template = mlreportgen.report.Report.createTemplate...  
('mytemplates/myReportTemplate','html');
```

After you modify the template, you can use it by setting the `TemplatePath` property of the `m!reportgen.report.Report` object.

## **See Also**

`m!reportgen.report.Report`

### **Topics**

“Modify Styles in a Microsoft Word Template” on page 13-126

“Modify Styles in HTML Templates” on page 13-137

“Modify Styles in PDF Templates” on page 13-138

### **Introduced in R2017b**

# mlreportgen.report.Report.customizeReport

**Class:** mlreportgen.report.Report

**Package:** mlreportgen.report

Create class derived from Report class

## Syntax

```
mlreportgen.report.Report.customizeReport(path)
```

## Description

`mlreportgen.report.Report.customizeReport(path)` creates a class definition file that defines a subclass of the `mlreportgen.report.Report` class. The new file is created at the location specified by `path`. You can use this file as a starting point for defining a custom report class.

## Input Arguments

**path — Class definition file path**

string | character array

Class definition file path, specified as a string or character array. The path specifies the location and name of the new class definition file.

## Examples

### Create Report Subclass

Create a new subclass named `package.MyClassB` of `mlreportgen.report.Report`.

```
mlreportgen.report.Report.customizeReport("+package/@MyClassB")
```

## See Also

`mlreportgen.report.Report`

**Introduced in R2017b**

## fill

**Class:** `mlreportgen.report.Report`

**Package:** `mlreportgen.report`

Fill report template holes

### Syntax

```
fill(report)
```

### Description

`fill(report)` fills each hole in the template of this report. Holes are filled with the value of the `report` property that has the same name as the hole. This method assumes that `report` is a subclass of `mlreportgen.report.Report`, which defines the holes and the properties that define how to fill the holes.

### Input Arguments

**report — Report**

`report` object

Report, specified as a report object. To create the `report` input, use `mlreportgen.report.Report`.

### See Also

`mlreportgen.report.Report`

**Introduced in R2017b**

# generateFileName

**Class:** mlreportgen.report.Report

**Package:** mlreportgen.report

Generate temporary report file name

## Syntax

```
fname = generateFileName(report)
```

```
fname = generateFileName(report,ext)
```

## Description

`fname = generateFileName(report)` returns a path string usable as the path of a file in the temporary folder of the report. You can use `generateFileName` to generate names for files that are stored temporarily until the report is closed.

`fname = generateFileName(report,ext)` uses the specified file extension.

## Input Arguments

### **report** — Report

report object

Report, specified as a report object. To create the report input, use `mlreportgen.report.Report`.

### **ext** — File extension

string

File extension, specified as an image file extension.

Example: `generateFileName(rpt,'jpg')`

## Output Arguments

### **fname** — File name

string

File name for temporary file, specified as a string.

## See Also

`mlreportgen.report.Report`

**Introduced in R2017b**

## **mlreportgen.report.Report.getClassFolder**

**Class:** mlreportgen.report.Report

**Package:** mlreportgen.report

Report class definition file location

### **Syntax**

```
path = mlreportgen.report.Report.getClassFolder()
```

### **Description**

`path = mlreportgen.report.Report.getClassFolder()` returns the path of the folder that contains the report class definition file.

### **Output Arguments**

**path** — Report class definition file location

character array

Report class definition file location, returned as a character array.

### **See Also**

mlreportgen.report.Report

**Introduced in R2017b**

# getContext

**Class:** mlreportgen.report.Report

**Package:** mlreportgen.report

Get report context value

## Syntax

```
cvalue = getContext(report, key)
```

## Description

`cvalue = getContext(report, key)` gets the report context value specified by the key. Use this method to retrieve report context information you have set previously using the `setContext` method. The context values and keys are stored in a `containers.Map` object.

## Input Arguments

### report — Report

report object

Report, specified as a report object. To create the report input, use `mlreportgen.report.Report`.

### key — Key associated with context value

numeric, real scalar | string | cell array

Key associated with context value, specified as a numeric, real scalar, string, or cell array.

## Output Arguments

### cvalue — Context value

any data type

Context value associated with specified key, returned as the value for that key.

## See Also

`mlreportgen.report.Report` | `containers.Map`

**Introduced in R2017b**

## getReportLayout

**Class:** `mlreportgen.report.Report`

**Package:** `mlreportgen.report`

Current page layout of report

### Syntax

```
pglayout = getReportLayout(report)
```

### Description

`pglayout = getReportLayout(report)` returns the current page layout of this report. Use this method in the `getImpl` method of a custom reporter to get the report page layout.

### Input Arguments

**report — Report**

report object

Report, specified as a report object. To create the report input, use `mlreportgen.report.Report`.

### Output Arguments

**pglayout — Current page layout**

page layout object | []

Current page layout, returned as a page layout object. This method returns the current layout whether the reporter is added directly to the report or indirectly as chapter content. The returned value depends on the report type.

- `pdf` — Returns an `mlreportgen.dom.PDFPageLayout`
- `docx` — Returns an `mlreportgen.dom.DOCXPageLayout`
- `HTML` — Returns [] because page layouts do not apply to HTML reports

### See Also

`mlreportgen.dom.PDFPageLayout` | `mlreportgen.dom.DOCXPageLayout`

**Introduced in R2018a**



# getTempPath

**Class:** `mlreportgen.report.Report`

**Package:** `mlreportgen.report`

Path of report temporary directory

## Syntax

```
path = getTempPath(report)
```

## Description

`path = getTempPath(report)` returns the path of the folder used for storing temporary files needed to generate the report.

## Input Arguments

**report — Report**

report object

Report, specified as a report object. To create the report input, use `mlreportgen.report.Report`.

## Output Arguments

**path — Path of the temporary folder**

string

Full path of the report temporary folder, returned as a string. By default the report temporary folder is a subset of your temporary folder. In debug mode, the report temporary folder is a subfolder of the report folder.

## See Also

`mlreportgen.report.Report`

**Introduced in R2018a**

## isdocx

**Class:** mlreportgen.report.Report

**Package:** mlreportgen.report

Check if Word report

### Syntax

```
typematch = isdocx(report)
```

### Description

`typematch = isdocx(report)` checks whether the report is a Microsoft Word report, which has a docx extension.

### Input Arguments

**report — Report**

report object

Report, specified as a report object. To create the report input, use `mlreportgen.report.Report`.

### Output Arguments

**typematch — Whether report is a Word report**

1 | 0

Whether report is a Microsoft Word report, returned as 1 or 0. If `typematch` is 1, the report is a Word report. Otherwise, `typematch` is 0.

### See Also

`mlreportgen.report.Report` | `ishtml` | `ishtmlfile` | `ispdf`

**Introduced in R2017b**

# ishtml

**Class:** `mlreportgen.report.Report`

**Package:** `mlreportgen.report`

Check if multifile HTML report

## Syntax

```
typematch = ishtml(report)
```

## Description

`typematch = ishtml(report)` checks whether the report is a multifile HTML report, which has an `htm` extension.

## Input Arguments

**report — Report**

report object

Report, specified as a report object. To create the report input, use `mlreportgen.report.Report`.

## Output Arguments

**typematch — Whether report is an HTML report**

1 | 0

Whether report is a multifile HTML report, returned as 1 or 0. If `typematch` is 1, the report is an HTML report. Otherwise, `typematch` is 0.

## See Also

`mlreportgen.report.Report` | `isdocx` | `ishtmlfile` | `ispdf`

**Introduced in R2017b**

## ishtmlfile

**Class:** `mlreportgen.report.Report`

**Package:** `mlreportgen.report`

Check if single-file HTML report

### Syntax

```
typematch = ishtmlfile(report)
```

### Description

`typematch = ishtmlfile(report)` checks whether the report is a single-file HTML report, which has an `html` extension.

### Input Arguments

**report — Report**

report object

Report, specified as a report object. To create the report input, use `mlreportgen.report.Report`.

### Output Arguments

**typematch — Whether report is a single-file HTML report**

1 | 0

Whether report is a single-file HTML report, returned as 1 or 0. If `typematch` is 1, the report is a single-file HTML report. Otherwise, `typematch` is 0.

### See Also

`mlreportgen.report.Report` | `isdocx` | `ishtml` | `ispdf`

**Introduced in R2017b**

# ispdf

**Class:** `mlreportgen.report.Report`

**Package:** `mlreportgen.report`

Check if PDF report

## Syntax

```
typematch = ispdf(report)
```

## Description

`typematch = ispdf(report)` checks whether the report is a PDF report, which has a `pdf` extension.

## Input Arguments

**report — Report**

report object

Report, specified as a report object. To create the report input, use `mlreportgen.report.Report`.

## Output Arguments

**typematch — Whether report is a PDF report**

1 | 0

Whether report is a PDF report, returned as 1 or 0. If `typematch` is 1, the report is a PDF report. Otherwise, `typematch` is 0.

## See Also

`mlreportgen.report.Report` | `isdocx` | `ishtml` | `ishtmlfile`

**Introduced in R2017b**

## open

**Class:** `mlreportgen.report.Report`

**Package:** `mlreportgen.report`

Opens the report

### Syntax

```
open(report)
```

### Description

`open(report)` opens the report. Using this method on a report that is already open or on a closed report causes an error. Use the `append` method to open the report if it is not already open. You generally use the `open` method directly only in an `open` method defined in a custom report class.

### Input Arguments

**report — Report**

report object

Report, specified as a report object. To create the report input, use `mlreportgen.report.Report`.

### See Also

`mlreportgen.report.Report` | `append` | `close` | `fill` | `rptview`

**Introduced in R2017b**

# rptview

**Class:** mlreportgen.report.Report

**Package:** mlreportgen.report

Open generated report file in viewer

## Syntax

```
rptview(report)
```

## Description

`rptview(report)` opens the generated report file in a viewer. The viewer used for displaying the report depends on the report type. For example, a report of type HTML opens in the default HTML browser.

To open a Word document on a Linux or Macintosh platform, `rptview` calls the `soffice` command. On Linux, Apache OpenOffice or LibreOffice must be installed. On Macintosh, Apache OpenOffice must be installed in the `/Applications` folder.

## Input Arguments

### **report** — Report

report object

Report, specified as a report object. To create the `report` input, use `mlreportgen.report.Report`.

## Tips

Calling `rptview` with an argument value other than an `mlreportgen.report.Report` object invokes the `rptview` function.

For example, in the following code, the first `rptview` call invokes the `rptview` method. The second `rptview` call invokes the `rptview` function.

```
import mlreportgen.report.*
rpt = Report('myReport','pdf');
append(rpt, 'Hello World');
close(rpt);
% Invokes rptview method
rptview(rpt);
% Invokes rptview function
rptview('myReport.pdf');
```

## See Also

`mlreportgen.report.Report` | `append` | `open`

**Introduced in R2017b**



# setContext

**Class:** `mlreportgen.report.Report`

**Package:** `mlreportgen.report`

Set report context value

## Syntax

```
setContext(report, key, cvalue)
```

## Description

`setContext(report, key, cvalue)` stores the context value (`cvalue`) specified by the `key` on the `report` object. You can then use the `getContext` method to retrieve the value. The context values and keys are stored in a `containers.Map` object.

## Input Arguments

### **report** — Report

report object

Report, specified as a report object. To create the `report` input, use `mlreportgen.report.Report`.

### **key** — Key to associate with context value

numeric, real scalar | string | cell array

Key to associate with context value, specified as a numeric, real scalar, string, or cell array.

### **cvalue** — Context value

any data type

Context value associated with specified key, specified as any type of value.

## See Also

`mlreportgen.report.Report` | `getContext` | `containers.Map`

**Introduced in R2017b**

## copy

**Class:** `mlreportgen.report.Reporter`

**Package:** `mlreportgen.report`

Create copy of reporter object and make deep copies of certain property values

### Syntax

```
reporterObj2 = copy(reporterObj1)
```

### Description

`reporterObj2 = copy(reporterObj1)` returns a copy of the specified reporter object. The returned copy contains a “deep copy” on page 11-291 of any property value of `reporterObj1` that references a reporter, DOM object, or `mlreportgen.report.ReporterLayout` object. As a result, the corresponding property value in `reporterObj2` refers to a new, independent object. You can modify the properties of the original or new object without affecting the other object.

### Input Arguments

**reportObj1 — Reporter to copy**

reporter object

Reporter to copy, specified as an object of a “reporter class” on page 11-291.

### Output Arguments

**reportObj2 — Copy of reporter**

reporter object

Copy of reporter, returned as an object of a “reporter class” on page 11-291.

### Examples

#### Copy a Reporter Object

This example copies a `MATLABVariable` reporter to show the effect of a deep copy operation on a reporter property. Modifying a property of the `Text` object in the `TextFormatter` property of the copy of the `MATLABVariable` object does not affect the original `MATLABVariable` object.

```
import mlreportgen.report.*
```

```
obj1 = MATLABVariable;
```

The `Bold` property of the `Text` object referenced by the `TextFormatter` property is empty.

```
obj1.TextFormatter.Bold
```

```
ans =  
[]
```

Copy the `MATLABVariable` object. In the copy, set the `Bold` property of the `Text` object referenced by the `TextFormatter` property to `true`.

```
obj2 = copy(obj1);  
obj2.TextFormatter.Bold = true;
```

In the original `MATLABVariable` object, the `Bold` property of the object referenced by the `TextFormatter` property is still empty.

```
obj1.TextFormatter.Bold  
ans =  
[]
```

## More About

### reporter class

A reporter class is a Report API class that is a subclass of the `mlreportgen.report.ReporterBase` class, which is an undocumented, internal class.

### deep copy

To make a deep copy of a handle object, the copy operation recursively copies property values that are handles to objects so that all of the underlying data is copied. By contrast, with a shallow copy, the copy operation copies the handle. The underlying data is not copied. When you copy a reporter, the copy operation makes a deep copy of any property value that is a reporter object, an `mlreportgen.report.ReporterLayout` object, or a DOM object.

## See Also

`matlab.mixin.Copyable` | `mlreportgen.report.Reporter`

### Topics

“Implement Copy for Handle Classes”

### Introduced in R2020a

## mlreportgen.report.Reporter.createTemplate

**Class:** mlreportgen.report.Reporter

**Package:** mlreportgen.report

Create reporter template

### Syntax

```
template = mlreportgen.report.Reporter.createTemplate(templatePath,type)
```

### Description

`template = mlreportgen.report.Reporter.createTemplate(templatePath,type)` creates a copy of the `mlreportgen.report.Reporter` template for the report type specified by `type` at the location specified by `templatePath`. You can use the copied template as a starting point to design a custom reporter template for your report.

### Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

"html" | "html-file" | "docx" | "pdf"

Type of template, specified as "html", "html-file", "docx", or "pdf".

### Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

### Examples

#### Create a Reporter Template

Create a copy of the HTML template for the `mlreportgen.report.Reporter` class and save it with the name `myTemplate` in the `mytemplates` folder.

```
template = mlreportgen.report.Reporter.createTemplate...  
    ('mytemplates/myTemplate','html');
```

After you modify the template, you can use it by setting the `TemplateSrc` property of a reporter to the path of the template file.

**See Also**

`mlreportgen.report.Reporter`

**Introduced in R2019a**

## mlreportgen.report.Reporter.customizeReporter

**Class:** mlreportgen.report.Reporter

**Package:** mlreportgen.report

Create class derived from Reporter class

### Syntax

```
reporter = mlreportgen.report.Reporter.customizeReporter(classpath)
```

### Description

`reporter = mlreportgen.report.Reporter.customizeReporter(classpath)` creates a reporter class definition file that defines a subclass of `mlreportgen.report.Reporter` at the location specified by `classpath`. The method also copies the default reporter templates to the `resources/templates` subfolder of the folder that contains the class definition file. You can use the class definition file as a starting point to design a custom reporter class.

### Input Arguments

**classpath — Path and name of new class definition file**

string scalar | character vector

Path and name of new class definition file, specified as a string scalar or character vector.

You can specify a relative path or an absolute path. For example, this code creates `MyClass.m` in the subfolder `myFolder` of the current folder.

```
mlreportgen.report.Reporter.customizeReporter("myFolder/MyClass")
```

To create the reporter class in a class folder, precede the class name with the `@` character. Do not specify the `.m` extension. For example, this code creates `MyClass.m` in the subfolder `myFolder/@MyClass` in the current folder.

```
mlreportgen.report.Reporter.customizeReporter("myFolder/@MyClass")
```

See “Folders Containing Class Definitions”.

To create the reporter class in a class package, precede the folder name with the `+` character. For example, this code creates a reporter in the `myOrg` package folder in the current folder.

```
mlreportgen.report.Reporter.customizeReporter("+myOrg/@MyClass")
```

### Output Arguments

**reporter — Path and file name of new reporter class**

string scalar

Path and file name of the new reporter class, returned as a string scalar.

## Examples

### Create Custom Reporter

Create a custom reporter, `myReporter`, and the associated default templates in the subfolder `MyFolder` of the current working folder.

```
mlreportgen.report.Reporter.customizeReporter("MyFolder/myReporter")
```

```
ans =
```

```
    "MyFolder\myReporter.m"
```

### See Also

`mlreportgen.report.Reporter`

### Topics

"What Is a Reporter?" on page 1-3

"Define a New Reporter" on page 1-8

### Introduced in R2019a

## **mlreportgen.report.Reporter.getClassFolder**

**Class:** mlreportgen.report.Reporter

**Package:** mlreportgen.report

Get location of folder containing mlreportgen.report.Reporter class definition file

### **Syntax**

```
path = mlreportgen.report.Reporter.getClassFolder()
```

### **Description**

path = mlreportgen.report.Reporter.getClassFolder() returns the path of the folder that contains the mlreportgen.report.Reporter class definition file.

### **Output Arguments**

**path** — mlreportgen.report.Reporter class definition file location

character vector

mlreportgen.report.Reporter class definition file location, returned as a character vector.

### **Examples**

#### **Get Reporter Class Folder**

Get the location of the folder that contains the mlreportgen.report.Reporter class definition file.

```
path = mlreportgen.report.Reporter.getClassFolder()
```

### **See Also**

mlreportgen.report.Reporter

**Introduced in R2019a**



# getImpl

**Class:** `mlreportgen.report.Reporter`

**Package:** `mlreportgen.report`

Get implementation of reporter

## Syntax

```
impl = getImpl(reporter, report)
```

## Description

`impl = getImpl(reporter, report)` returns the DOM object used to implement this reporter in the specified report. The implementation object can help you debug report generation problems.

## Input Arguments

### reporter — Reporter

`mlreportgen.report.Reporter` object | object of subclass of `mlreportgen.report.Reporter`

Reporter object, specified as an object of `mlreportgen.report.Reporter` or an object of a subclass of `mlreportgen.report.Reporter`.

Example: `mlreportgen.report.TitlePage` for a title page reporter

### report — Report

report object

Report, specified as a report object. To create the report input, use `mlreportgen.report.Report`.

## Output Arguments

### impl — Implementation object

DOM object

Implementation object, returned as a DOM object. The DOM object is usually an `mlreportgen.dom.DocumentPart` type object.

## Examples

### Get Reporter Implementation

This example shows how to use `getImpl` to obtain the DOM object used to create a `TitlePage` reporter. You can use the `getImpl` method with any type of reporter.

```
import mlreportgen.report.*  
rpt = Report('My Report');  
tp = TitlePage;
```

```
tp.Title = 'Data Summary';  
impl = getImpl(tp,rpt)
```

### **See Also**

mlreportgen.report.Reporter

**Introduced in R2017b**

# mlreportgen.report.RptFile.createTemplate

**Class:** mlreportgen.report.RptFile

**Package:** mlreportgen.report

Create Report Explorer-based (RptFile) reporter template

## Syntax

```
template = mlreportgen.report.RptFile.createTemplate(templatePath,type)
```

## Description

`template = mlreportgen.report.RptFile.createTemplate(templatePath,type)` creates a copy of the default Report Explorer-based reporter (RptFile) template specified by `type` at the `templatePath` location. You can use the copied template as a starting point to design a custom RptFile reporter template for your report.

## Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

"html" | "html-file" | "docx" | "pdf"

Type of template, specified as "html", "html-file", "docx", or "pdf".

## Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdf.tx`.

## Examples

### Create Report Explorer-based Reporter Template

Before you run this example, create a Report Explorer setup file (see “Create a New Setup File” on page 4-4) or use an existing setup file. In this example, the setup file is named `mysetupfile.rpt`. Then, create a copy of the default HTML RptFile template and edit it as desired. The copied template file in this example is named `myrptfile.htm.tx` and is saved in a folder named `mytemplates`. To use the new template for the RptFile reporter, assign its path to the RptFile `TemplateSrc` property.

```
import mlreportgen.report.*
rpt = Report('My Report','html');
rptfile = RptFile('mysetupfile.rpt');
template = RptFile.createTemplate('mytemplates\myrptfile','html');
rptfile.TemplateSrc = template;
```

## See Also

[mlreportgen.report.RptFile](#) | [mlreportgen.report.Reporter](#) | [mlreportgen.report.Report](#)

## Topics

“Modify Styles in a Microsoft Word Template” on page 13-126

“Modify Styles in HTML Templates” on page 13-137

“Modify Styles in PDF Templates” on page 13-138

## Introduced in R2019a

# mlreportgen.report.RptFile.customizeReporter

**Class:** mlreportgen.report.RptFile

**Package:** mlreportgen.report

Create custom Report Explorer-based reporter class

## Syntax

```
reporter = mlreportgen.report.RptFile.customizeReporter(classpath)
```

## Description

`reporter = mlreportgen.report.RptFile.customizeReporter(classpath)` creates a Report Explorer-based reporter (`RptFile`) class definition file that is a subclass of `mlreportgen.report.RptFile`. The file is created at the specified `classpath` location. The `RptFile.customizeReporter` method also copies the default `RptFile` templates to the `<classpath>/resources/template` folder. You can use the new class definition file as a starting point to design a custom Report Explorer-based reporter class for your report.

## Input Arguments

**classpath** — Location of custom Report Explorer-based reporter class

current working folder (default) | string | character array

Location of custom Report Explorer-based reporter class, specified as a string or character array. The `classpath` argument also supports specifying a folder with `@` before the class name.

## Output Arguments

**reporter** — Report Explorer-based reporter path

string

Report Explorer-based reporter path, returned as a string specifying the path to the derived report class file.

## Examples

### Create Custom Report Explorer-based Reporter

Create a custom Report Explorer-based reporter and its associated default templates. The derived class file is created at the specified path relative to the current working folder. In this example, the path to the `MyRptExplRptr.m` class file is `<current working folder>/new_rptexpl_rptr/@MyRptExplRptr/MyRptExplRptr.m`. The default `RptFile` templates are in the `<current working folder>/new_rptexpl_rptr/@RptExplRptr/resources/templates` folder.

```
import mlreportgen.report.*
RptFile.customizeReporter('new_rptexpl_rptr/@MyRptExplRptr');
```

After editing this new class file, you can use it as your `RptFile` reporter.

```
rptr = MyRptExplRptr();
```

### **See Also**

`mlreportgen.report.RptFile` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Report`

**Introduced in R2019a**

# mlreportgen.report.RptFile.getClassFolder

**Class:** mlreportgen.report.RptFile

**Package:** mlreportgen.report

Report Explorer-based reporter class definition file location

## Syntax

```
path = mlreportgen.report.RptFile.getClassFolder()
```

## Description

`path = mlreportgen.report.RptFile.getClassFolder()` returns the path of the folder that contains the Report Explorer-based reporter class definition file.

## Output Arguments

**path — Report Explorer-based reporter class definition file location**

character array

Report Explorer-based reporter class definition file location, returned as a character array.

## See Also

mlreportgen.report.RptFile | mlreportgen.report.Reporter |

mlreportgen.report.Report

**Introduced in R2019a**

## add

**Class:** `mlreportgen.report.Section`

**Package:** `mlreportgen.report`

(Not recommended) Add content to section

### Syntax

```
add(section,content)
```

### Description

---

**Note** `add` is not recommended. Use `append` instead. See “Compatibility Considerations” on page 11-305.

---

`add(section,content)` adds the specified content to the specified section.

### Input Arguments

**section — Section of the report**

`mlreportgen.report.Section` object

Section of the report, specified as an `mlreportgen.report.Section` object.

**content — Content to add to section**

Report API reporter | DOM object | MATLAB built-in object | cell array of objects

Content to add to the section, specified as one of these values:

- Report API reporter
- DOM object
- Built-in MATLAB object (most built-in MATLAB objects can be added to a `Section` reporter)
- Cell array of objects that can be added individually to a section

### Examples

#### Add Content to Report Section

Add content to a paragraph and add the paragraph to a section of a chapter. To add the content to the paragraph, you must use `append` because a paragraph is a DOM API object. To add the paragraph to the section, the section to the chapter, and the chapter to the report, this example uses `add`. Starting in R2020b, you can use `append` instead of `add`. See “Compatibility Considerations” on page 11-268.

```
import mlreportgen.dom.*
import mlreportgen.report.*
```



```

rpt = Report("My Report");
ch = Chapter("My Chapter");
s = Section("My Section");
p = Paragraph("My paragraph content ");
append(p, "and some more content.");
add(s,p);
add(ch,s);
add(rpt,ch);

close(rpt);
rptview(rpt);

```

## Compatibility Considerations

### add method is not recommended

*Not recommended starting in R2020b*

Starting in R2020b, use the `append` method instead of the `add` method to add content to objects of these Report API classes:

- `mlreportgen.report.Report`
- `mlreportgen.report.Chapter`
- `mlreportgen.report.Section`

To add content to a DOM API object, such as an `mlreportgen.dom.Paragraph` object, continue to use the `append` method of the DOM object. The advantage of using `append` to add content to Report API objects is that you use the same method name as you use to add content to DOM API objects.

There are no plans to remove the `add` methods of the `Report`, `Chapter`, or `Section` classes. Report API programs that use the `add` methods will continue to run.

To update existing code, replace the method name `add` with `append` as shown by the examples in the table.

Not Recommended	Recommended
<pre> import mlreportgen.report.* import mlreportgen.dom.*  rpt = Report("My Report","pdf"); ch = Chapter("My Chapter"); sect = Section("My Section"); para = Paragraph("My Content "); add(para,"more Content"); add(sect,para); add(ch,sect); add(rpt,ch);  close(rpt); rptview(rpt); </pre>	<pre> import mlreportgen.report.* import mlreportgen.dom.*  rpt = Report("My Report","pdf"); ch = Chapter("My Chapter"); sect = Section("My Section"); para = Paragraph("My Content "); append(para,"more Content"); append(sect,para); append(ch,sect); append(rpt,ch);  close(rpt); rptview(rpt); </pre>

## See Also

`mlreportgen.report.Section` | `mlreportgen.report.Reporter` | `mlreportgen.report.Report` | `append`

**Topics**

“Add Content to a Report” on page 13-10

**Introduced in R2017b**

# append

**Class:** `mlreportgen.report.Section`

**Package:** `mlreportgen.report`

Add content to section

## Syntax

```
append(section,content)
```

## Description

`append(section,content)` adds the specified content to the specified section.

## Input Arguments

### **section** — Section of the report

`mlreportgen.report.Section` object

Section of the report, specified as an `mlreportgen.report.Section` object.

### **content** — Content to add to section

Report API reporter | DOM object | MATLAB built-in object | cell array of objects

Content to add to the section, specified as one of these values:

- Report API reporter
- DOM object
- Built-in MATLAB object (many built-in MATLAB objects can be added to a `Section` reporter)
- Cell array of objects that can be added individually to a section

## Examples

### **Add Content to a Section**

This example generates a report that has a title page and one chapter. The chapter contains two sections, each of which contains an image. The example uses `append` to add the content to the images, the images to the sections, the sections to the chapter, and the chapter and title page to the report.

Import the DOM and API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*  
import mlreportgen.dom.*
```

Create a report and add a title page to the report.

```
rpt = Report("My Report","pdf");  
append(rpt,TitlePage("Title","My Report"));
```

Create a chapter with two sections. Add an image to each section. Add the sections to the chapter and the chapter to the report.

```
ch = Chapter("Images");  
sect1 = Section("Boeing 747");  
append(sect1, Image("b747.jpg"));  
append(ch, sect1);  
sect2 = Section("Peppers");  
append(sect2, Image("peppers.png"));  
append(ch, sect2);  
append(rpt, ch);
```

Close and view the report.

```
close(rpt);  
rptview(rpt);
```

## See Also

`mlreportgen.report.Section`

## Topics

"Add Content to a Report" on page 13-10

**Introduced in R2020b**

# mlreportgen.report.Section.createTemplate

**Class:** mlreportgen.report.Section

**Package:** mlreportgen.report

Create section template

## Syntax

## Description

creates a copy of the default section reporter template specified by `type` at the location specified by `templatePath`. You can use the copied template as a starting point to design a custom section template for your report.

## Input Arguments

### **templatePath** — Path and file name of new template

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

### **type** — Type of template

"html" | "html-file" | "docx" | "pdf"

Type of template, specified as "html", "html-file", "docx", or "pdf".

## Output Arguments

### **template** — Path and file name of template copy

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdfx`.

## Examples

### Create Section Template

Before you run this example, create a copy of the default HTML Section template in the `mytemplates` folder. Name the copied template `mySection.htmx`. Edit the template as desired. To use the new template for the section, assign its path to the `Section.TemplateSrc` property.

```
import mlreportgen.report.*;
rpt = Report('My Report','html');
sec = Section();
sec.Title = 'Data Summary';
```

```
template = Section.createTemplate('mytemplates\mySection', 'html');  
sec.TemplateSrc = template;
```

### See Also

mlreportgen.report.Section | mlreportgen.report.Reporter |  
mlreportgen.report.Report

### Topics

“Modify Styles in a Microsoft Word Template” on page 13-126

“Modify Styles in HTML Templates” on page 13-137

“Modify Styles in PDF Templates” on page 13-138

### Introduced in R2017b

# mlreportgen.report.Section.customizeReporter

**Class:** mlreportgen.report.Section

**Package:** mlreportgen.report

Create custom section reporter class

## Syntax

```
reporter = mlreportgen.report.Section.customizeReporter(classpath)
```

## Description

`reporter = mlreportgen.report.Section.customizeReporter(classpath)` creates a section class definition file that is a subclass of `mlreportgen.report.Section`. The file is created at the specified `classpath` location. The `Section.customizeReporter` method also copies the default section templates to the `<classpath>/resources/template` folder. You can use the new class definition file as a starting point to design a custom section class for your report.

## Input Arguments

**classpath — Location of custom section class**

current working folder (default) | string | character array

Location of custom section class, specified as a string or character array. The `classpath` argument also supports specifying a folder with `@` before the class name.

## Output Arguments

**reporter — Section reporter path**

string

Section reporter path, returned as a string specifying the path to the derived report class file.

## Examples

### Create Custom Section Reporter

Create a custom section reporter and its associated default templates. The derived class file is created at the specified path relative to the current working folder. In this case, the path to the `MySection.m` class file is `<current working folder>/newSection/@MySection/MySection.m`. The default title page templates are in the `<current working folder>/newSection/@MySection/resources/templates` folder.

```
import mlreportgen.report.*
Section.customizeReporter('newSection/@MySection');
```

After editing this new class file, you can use it as your title page reporter.

```
tp = MySection();
```

**See Also**

`mlreportgen.report.Section` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Report`

**Introduced in R2017b**



# mlreportgen.report.Section.getClassFolder

**Class:** mlreportgen.report.Section

**Package:** mlreportgen.report

Section class definition file location

## Syntax

```
path = mlreportgen.report.Section.getClassFolder()
```

## Description

`path = mlreportgen.report.Section.getClassFolder()` returns the path of the folder that contains the section reporter class definition file.

## Output Arguments

**path** — Section class definition file location

character array

Section class definition file location, returned as a character array.

## See Also

mlreportgen.report.Section | mlreportgen.report.Reporter |

mlreportgen.report.Report

**Introduced in R2017b**

## getTitleReporter

**Class:** `mlreportgen.report.Section`

**Package:** `mlreportgen.report`

Create a section title reporter

### Syntax

```
reporter = getTitleReporter(section)
```

### Description

`reporter = getTitleReporter(section)` creates an `mlreportgen.report.SectionTitle` reporter that is partially configured to:

- Format the value of the `Title` property of the `mlreportgen.report.Section` reporter specified by `section`.
- Fill the `Title` hole in the `Section` reporter template with the formatted title.

The `SectionTitle` reporter `getImpl` method uses the `mlreportgen.report.Section.getTitleReporter` method to format inline title content. You can use the `mlreportgen.report.Section.getTitleReporter` method to customize inline title format as follows:

- 1 Invoke the `mlreportgen.report.Section.getTitleReporter` method to get a default `SectionTitle` reporter.
- 2 Customize the `SectionTitle` reporter properties. For example, specify a template source that contains customized title templates.
- 3 Set the `Title` property of the `Section` reporter to the customized `SectionTitle` reporter.

You can customize the title format by changing the default values of these `SectionTitle` reporter properties:

- `TemplateSrc`

This property is set by default to the `Section` reporter template. The template library of this template contains default section title templates named:

- `SectionTitle1`, `SectionTitle2`, and so on, for unnumbered titles
- `SectionNumberedTitle1`, `SectionNumberedTitle2`, and so on, for hierarchically numbered titles.

You can set this property to the source of a template file that contains custom definitions of these templates.

- `TemplateName`

This property is set by default to `'SectionNumberedTitle'` if the `Section` reporter `Numbered` property is `true` (the default). Otherwise, this property is set to `'SectionTitle'`. You do not need to change this setting if your custom template library customizes the definitions of the

standard title templates, but not their names. For example, if your template library contains a template named `SectionNumberedTitle1` with a customized version of the standard definition for `SectionNumberedTitle1`, you do not need to change the value of this property.

- `OutlineLevel`

This property is empty by default. The `Section` reporter `getImpl` method, which invokes the `getTitleReporter` method by default, sets this property to a section level when the `Section` reporter is added to the report. The `SectionTitle` reporter then appends the value of the `OutlineLevel` property to the value of the `TemplateName` property to create the full name of the `SectionTitle` template for the section level, for example, `SectionNumberedTitle2` for a second-level subsection.

- `Content`

This property is set by default to the value of the `Section` reporter `Title` property.

- `NumberPrefix`

This property is set by default to `[]`.

- `NumberSuffix`

This property is set to `' . '` if the `Section Numbered` property is `true` (the default). Otherwise, this property is set to `[]`.

- `Translations`

This property is not set by default. Set this property to localize the section title number prefix and suffix title content.

## Input Arguments

### **section** — Section of the report

`m1reportgen.report.Section` object

Section of the report, specified as an `m1reportgen.report.Section` object.

## Output Arguments

### **reporter** — Section title reporter

`m1reportgen.report.SectionTitle` object

Section title reporter, returned as an `m1reportgen.report.SectionTitle` object.

## Examples

### **Use a Custom Section Title Template**

The following example shows how to use a custom template to generate the title of a level one section in a report.

Suppose that `MyCustomTemplate.pdf` contains a customized version of the standard template `SectionNumberedTitle1`.

To use the custom template:

- 1 Get the `SectionTitle` reporter for the section.
- 2 Set the `TemplateSrc` property of the reporter to `MyCustomTemplate.pdf` and set the `OutlineLevel` property to the level of the section.
- 3 Set the `Title` property of the Section reporter to the `SectionTitle` reporter.

For example:

```
import mlreportgen.report.*  
  
rpt = Report("myrpt", "pdf");  
  
sec = Section("Title", "Introduction");  
titleRptr = getTitleReporter(sec);  
titleRptr.TemplateSrc = "MyCustomTemplate.pdf";  
titleRptr.OutlineLevel = 1;  
sec.Title = titleRptr;  
add(rpt, sec);  
close(rpt);
```

### See Also

`mlreportgen.report.Section` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Report`

**Introduced in R2017b**

# mlreportgen.report.Section.number

**Class:** mlreportgen.report.Section

**Package:** mlreportgen.report

Set section numbering

## Syntax

```
mlreportgen.report.Section.number(report,numbering)
```

## Description

mlreportgen.report.Section.number(report,numbering) indicates whether to include numbers before report section titles. Report sections are numbered consecutively by default.

## Input Arguments

### report — Report

report object

Report, specified as a report object. To create the report input, use mlreportgen.report.Report.

### numbering — Option to number sections

true (default) | false

Choice to number report sections, specified as a logical. If numbering is true, the report sections are numbered consecutively. If numbering is false, numbers are not included for report sections. You can use the Numbered property of a section to override the numbering for that section.

## Examples

### Turn Off Section Numbering

```
import mlreportgen.report.*
import mlreportgen.dom.*
rpt = Report('My Report','pdf');
add(rpt,TitlePage('Title','My Report',...
    'Author','MathWorks'));
add(rpt,TableOfContents);
add(rpt,Chapter('Title','Boeing 747',...
    'Content','This report describes the Boeing 747.));
mlreportgen.dom.KeepWithNext(1);
add(rpt,Section('Title','Image of Boeing 747',...
    'Content',Image(which('b747.jpg')),'Numbered',0));
close(rpt);
rptview(rpt);
```

**See Also**

`mlreportgen.report.Section` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Report`

**Introduced in R2017b**

# mlreportgen.report.SummaryTable.createTemplate

**Package:** mlreportgen.report

Create summary table reporter template

## Syntax

```
template = mlreportgen.report.SummaryTable.createTemplate(templatePath,type)
```

## Description

`template = mlreportgen.report.SummaryTable.createTemplate(templatePath,type)` creates a copy of the `mlreportgen.report.SummaryTable` reporter template for the report type specified by `type` at the location specified by `templatePath`.

## Examples

### Create Summary Table Reporter Template

Create a copy of the HTML template for the `mlreportgen.report.SummaryTable` class and save it with the name `myTemplate` in the `mytemplates` folder.

```
template = mlreportgen.report.SummaryTable.createTemplate("mytemplate/myTemplate",'html')
```

After you modify the template, use it by setting the `TemplateSrc` property of a summary table reporter to the path of the new template file.

## Input Arguments

### templatePath — Path and file name of new template

string scalar (default) | character vector

Path and file name of the new template, specified as a character vector or a string scalar.

### type — Type of template

"html" | "html-file" | "docx" | "pdf"

Type of template, specified as "html", "html-file", "docx", or "pdf".

## Output Arguments

### template — Path and file name of template copy

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

**See Also**

`mlreportgen.report.SummaryTable`

**Introduced in R2022a**



# mlreportgen.report.SummaryTable.customizeReporter

**Package:** mlreportgen.report

Create class derived from summary table reporter class

## Syntax

```
reporter = mlreportgen.report.SummaryTable.customizeReporter(classPath)
```

## Description

`reporter = mlreportgen.report.SummaryTable.customizeReporter(classPath)` creates a reporter class definition file that defines a subclass of `mlreportgen.report.SummaryTable` at the location specified by `classPath`. The method also copies the default reporter template to the `resources/templates` subfolder of the folder that contains the class definition file. Use the class definition file as a starting point to design a custom reporter class.

## Examples

### Create Custom Summary Table Reporter

Create a custom reporter and its associated default templates. This function creates the derived class file at the specified path relative to the current working folder.

In this case, the path to the `mySummaryTableReporter.m` class file is `<current_working_folder>/newSummaryTableReporter/@mySummaryTableReporter/mySummaryTableReporter.m`.

```
mlreportgen.report.SummaryTable.customizeReporter("mySummaryTable/@mySummaryTableReporter")
```

After editing this new class file, you can use it as your summary table reporter.

The default are in the `<current_working_folder>/newSummaryTableReporter/@mySummaryTableReporter/resources/templates` folder.

## Input Arguments

### **classPath** — Path and name of new class definition file

string scalar | character vector

Path and name of a new class definition file, specified as a string scalar or character vector.

Specify a relative path or an absolute path. For example, this code creates `mySummaryTableReporter.m` in the subfolder `sFolder1` of the current folder.

```
mlreportgen.report.SummaryTable.customizeReporter("sFolder1/mySummaryTableReporter")
```

To create the reporter class in a class folder, precede the class name with the @ character. Do not specify the .m extension. For example, this code creates myClass.m in the subfolder sFolder2/@myClass in the current folder.

```
slreportgen.report.SummaryTable.customizeReporter("sFolder2/@myClass")
```

See “Folders Containing Class Definitions”.

To create the reporter class in a package, precede the folder name with the + character. For example, this code creates a reporter in the myOrg package folder in the current folder.

```
slreportgen.report.SummaryTable.customizeReporter("+myOrg/@myClass")
```

## Output Arguments

**reporter** — Path and file name of the new reporter class

string scalar

Path and file name of the new reporter class, returned as a string scalar.

## See Also

mlreportgen.report.SummaryTable

**Introduced in R2022a**

# mlreportgen.report.SummaryTable.getClassFolder

**Package:** mlreportgen.report

Get location of folder containing mlreportgen.report.SummaryTable class definition file

## Syntax

```
path = mlreportgen.report.SummaryTable.getClassFolder()
```

## Description

path = mlreportgen.report.SummaryTable.getClassFolder() returns the path of the folder that contains the mlreportgen.report.SummaryTable class definition file.

## Examples

### Get Summary Table Reporter Class Folder

Get the location of the folder that contains the slreportgen.report.SummaryTable class definition file.

```
path = slreportgen.report.SummaryTable.getClassFolder()
```

This function returns the path as a character vector.

## Output Arguments

**path** — Class definition file location

character vector

slreportgen.report.SummaryTable class definition file location, returned as a character vector.

## See Also

mlreportgen.report.SummaryTable

**Introduced in R2022a**

# mlreportgen.report.TableOfContents.createTemplate

**Class:** mlreportgen.report.TableOfContents

**Package:** mlreportgen.report

Create table of contents section template

## Syntax

```
template = mlreportgen.report.TableOfContents.createTemplate(templatePath,  
type)
```

## Description

`template = mlreportgen.report.TableOfContents.createTemplate(templatePath, type)` creates a copy of the table of contents (TOC) reporter template specified by `type` at the location specified by `templatePath`. You can use the copied template as a starting point to design a custom TOC template for your report.

## Input Arguments

**templatePath** — Path and file name of new template

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type** — Type of template

"html" | "html-file" | "docx" | "pdf"

Type of template, specified as "html", "html-file", "docx", or "pdf".

## Output Arguments

**template** — Path and file name of template copy

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdf.tx`.

## Examples

### Create TOC Template

Before you run this example, create a copy of the default HTML `TableOfContents` template in the `mytemplates` folder. Name the copied template `myTOC.html.tx`. Edit the template as desired. To use the new template for the TOC, assign its path to the `TableOfContents.TemplateSrc` property.

```
import mlreportgen.report.*  
rpt = Report('My Report', 'html');  
toc = TableOfContents();  
template = TableOfContents.createTemplate('mytemplates\myTOC', 'html');  
toc.TemplateSrc = template;
```

## See Also

mlreportgen.report.TableOfContents | mlreportgen.report.Reporter |  
mlreportgen.report.Report

## Topics

“Modify Styles in a Microsoft Word Template” on page 13-126

“Modify Styles in HTML Templates” on page 13-137

“Modify Styles in PDF Templates” on page 13-138

## Introduced in R2017b

# mlreportgen.report.TableOfContents.customizeReporter

**Class:** mlreportgen.report.TableOfContents

**Package:** mlreportgen.report

Create custom table of contents reporter class

## Syntax

```
reporter = mlreportgen.report.TableOfContents.customizeReporter(classpath)
```

## Description

`reporter = mlreportgen.report.TableOfContents.customizeReporter(classpath)` creates a table of contents (TOC) class definition file that is a subclass of `mlreportgen.report.TableOfContents`. The file is created at the specified `classpath` location. The `TableOfContents.customizeReporter` method also copies the default TOC templates to the `<classpath>/resources/template` folder. You can use the class definition file as a starting point to design a custom table of contents class for your report.

## Input Arguments

**classpath — Location of custom TOC section class**

current working folder (default) | string | character array

Location of custom TOC section class, specified as a string or character array. The classpath argument also supports specifying a folder with @ before the class name.

## Output Arguments

**reporter — TOC section reporter path**

string

TOC Section reporter path, returned as the string specifying the path to the derived report class file.

## Examples

### Create Custom TOC Reporter

Create a custom TOC reporter and its associated default templates. The derived class file is created at the specified path relative to the current working folder. In this case, the path to the `MyTOC.m` class file is `<current working folder>/newTOC/@MyTOC/myTOC.m`. The default title page templates are in the `<current working folder>/newTOC/@MyTOC/resources/templates` folder.

```
import mlreportgen.report.*  
TableOfContents.customizeReporter('newTOC/@MyTOC');
```

After editing this new class file, you can use it as your TOC section reporter.

```
toc = MyTOC();
```

### **See Also**

[mlreportgen.report.TableOfContents](#) | [mlreportgen.report.Report](#) | [mlreportgen.report.Reporter](#)

**Introduced in R2017b**

## **mlreportgen.report.TableOfContents.getClassFolder**

**Class:** mlreportgen.report.TableOfContents

**Package:** mlreportgen.report

Table of contents class definition file location

### **Syntax**

```
path = mlreportgen.report.TableOfContents.getClassFolder()
```

### **Description**

path = mlreportgen.report.TableOfContents.getClassFolder() returns the path of the folder that contains the table of contents class definition file.

### **Output Arguments**

**path** — **Table of contents class definition file location**

character array

Table of contents class definition file location, returned as a character array.

### **See Also**

mlreportgen.report.TableOfContents | mlreportgen.report.Reporter | mlreportgen.report.Report

**Introduced in R2017b**



# getTitleReporter

**Class:** mlreportgen.report.TableOfContents

**Package:** mlreportgen.report

Get table of contents title reporter

## Syntax

```
reporter = getTitleReporter(toc)
```

## Description

`reporter = getTitleReporter(toc)` returns a reporter that the `TableOfContents` reporter uses to format the content specified in its `Title` property. Use this reporter to customize the alignment, position, and appearance of the title.

## Input Arguments

**toc — Table of contents of the report**

reporter object

Table of contents of the report, specified as a reporter object. To create the `toc` input, you use `mlreportgen.report.TableOfContents`.

## Output Arguments

**reporter — Table of contents title reporter**

reporter object

Table of contents title reporter, returned as a reporter object.

## Examples

### Use Nondefault Table of Contents Title Style

Assume that you want a style for your table of contents title that differs from the default. Before you run this example, create a template file named `MyTOCTemplate` and customize its `TableOfContentsTitle` style. Then, use the `getTitleReporter` method and the `TemplateSrc` property to use your template.

```
import mlreportgen.report.*
rpt = Report();
toc = TableOfContents();
toc.Title = 'Report Contents'
toc.Title = getTitleReporter(toc);
toc.Title.TemplateSrc = 'MyTOCTemplate';
add(rpt,toc)
```

**See Also**

`mlreportgen.report.TableOfContents` | `mlreportgen.report.Reporter`

**Introduced in R2017b**

# mlreportgen.report.TitlePage.createTemplate

**Class:** mlreportgen.report.TitlePage

**Package:** mlreportgen.report

Create title page template

## Syntax

```
template = mlreportgen.report.TitlePage.createTemplate(templatePath,type)
```

## Description

`template = mlreportgen.report.TitlePage.createTemplate(templatePath,type)` creates a copy of the default title page reporter template specified by `type` at the location specified by `templatePath`. You can use the copied template as a starting point to design a custom title page template for your report.

## Input Arguments

**templatePath — Path and file name of new template**

character vector | string scalar

Path and file name of the new template, specified as a character vector or string scalar.

**type — Type of template**

"html" | "html-file" | "docx" | "pdf"

Type of template, specified as "html", "html-file", "docx", or "pdf".

## Output Arguments

**template — Path and file name of template copy**

string scalar

Path and file name of the copy of the template, returned as a string scalar. The file name extension of the template is based on the specified template type. For example, if the `type` argument is 'pdf', the file name extension is `.pdftx`.

## Examples

### Create Title Page Template

Before you run this example, create a copy of the default HTML TitlePage template in the `mytemplates` folder. Name the copied template `myTitlePg.htm`. Edit the template as desired. To use the new template for the title page, assign its path to the `TitlePage.TemplateSrc` property.

```
import mlreportgen.report.*  
rpt = Report('My Report','html');  
tp = TitlePage();
```

```
tp.Title = 'Data Summary';  
template = TitlePage.createTemplate('mytemplates\myTitlePg','html');  
tp.TemplateSrc = template;
```

**See Also**

[mlreportgen.report.TitlePage](#) | [mlreportgen.report.Report](#) |  
[mlreportgen.report.Reporter](#) | [mlreportgen.report.TitlePage.customizeReporter](#)

**Topics**

“Modify Styles in a Microsoft Word Template” on page 13-126

“Modify Styles in HTML Templates” on page 13-137

“Modify Styles in PDF Templates” on page 13-138

**Introduced in R2017b**

# mlreportgen.report.TitlePage.customizeReporter

**Class:** mlreportgen.report.TitlePage

**Package:** mlreportgen.report

Create custom title page reporter class

## Syntax

```
reporter = mlreportgen.report.TitlePage.customizeReporter(classpath)
```

## Description

`reporter = mlreportgen.report.TitlePage.customizeReporter(classpath)` creates a title page class definition file that is a subclass of `mlreportgen.report.TitlePage`. The file is created at the specified `classpath` location. The `TitlePage.customizeReporter` method also copies the default title page templates to the `<classpath>/resources/template` folder. You can use the new class definition file as a starting point to design a custom title page class for your report. You can customize the title page class by, for example, adding new content holes and properties.

## Input Arguments

**classpath** — Location of custom title page class

current working folder (default) | string | character array

Location of custom title page class, specified as a string or character array. The `classpath` argument also supports specifying a folder with `@` before the class name.

## Output Arguments

**reporter** — Title page reporter path

string

Title page reporter path, returned as the string specifying the path to the derived report class file.

## Examples

### Create Custom Title Page Reporter

Create a custom title page reporter and its associated default templates. The derived class file is created at the specified path relative to the current working folder. In this case, the path to the `MyTitlePage.m` class file is `<current working folder>/newTitlePage/@MyTitlePage/MyTitlePage.m`. The default title page templates are in the `<current working folder>/newTitlePage/@MyTitlePage/resources/templates` folder.

```
import mlreportgen.report.*  
TitlePage.customizeReporter('newTitlePg/@MyTitlePage');
```

After editing this new class file, you can use it as your title page reporter.

```
tp = MyTitlePage();
```

### **See Also**

`mlreportgen.report.TitlePage` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Report` | `mlreportgen.report.TitlePage.createTemplate`

**Introduced in R2017b**

# getAuthorReporter

**Class:** mlreportgen.report.TitlePage

**Package:** mlreportgen.report

Get title page author reporter

## Syntax

```
reporter = getAuthorReporter(tp)
```

## Description

`reporter = getAuthorReporter(tp)` returns a reporter that the `TitlePage` reporter (`tp`) uses to format the value specified by its `Author` property.

## Input Arguments

**tp — Title page of the report**

reporter object

Title page of the report, specified as a title page reporter object. To create the `tp` input, you use `mlreportgen.report.TitlePage`.

## Output Arguments

**reporter — Title page author reporter**

reporter object

Title page author reporter, returned as a reporter object.

## Examples

### Use Nondefault Title Page Author Style

Create a style for your title page author that differs from the default style. Before you run this example, create a template file named `MyTitlePageTemplate` and customize its `TitlePageAuthor` style. Then, use the `getAuthorReporter` method and the `TemplateSrc` property to use your template.

```
import mlreportgen.report.*
rpt = Report();
tp = TitlePage();
tp.Author = 'John Smith';
tp.Author = getAuthorReporter(tp);
tp.Author.TemplateSrc = 'MyTitlePageTemplate';
add(rpt, tp);
```

**See Also**

`mlreportgen.report.TitlePage` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Report`

**Introduced in R2017b**



# mlreportgen.report.TitlePage.getClassFolder

**Class:** mlreportgen.report.TitlePage

**Package:** mlreportgen.report

Title page class definition file location

## Syntax

```
path = mlreportgen.report.TitlePage.getClassFolder()
```

## Description

`path = mlreportgen.report.TitlePage.getClassFolder()` returns the path of the folder that contains the title page class definition file.

## Output Arguments

**path** — Title page class definition file location

character array

Title page class definition file location, returned as a character array.

## See Also

mlreportgen.report.TitlePage | mlreportgen.report.Reporter |

mlreportgen.report.Report

**Introduced in R2017b**

## getImageReporter

**Class:** mlreportgen.report.TitlePage

**Package:** mlreportgen.report

Get title page image reporter

### Syntax

```
reporter = getImageReporter(tp)
```

### Description

`reporter = getImageReporter(tp)` returns a reporter that the `TitlePage` reporter (`tp`) uses to format the image specified by the `Image` property. You use `getImageReporter` to customize the image position and alignment.

### Input Arguments

**tp — Title page of the report**

reporter object

Title page of the report, specified as a title page reporter object. To create the `tp` input, you use `mlreportgen.report.TitlePage`.

### Output Arguments

**reporter — Title page image reporter**

reporter object

Title page image reporter, returned as a reporter object.

### Examples

#### Use Nondefault Title Page Image Style

Create a style for the image on your title page that differs from the default. Before you run this example, create a template file named `MyTitlePageTemplate` and customize its `TitlePageImage` style. Then, use `getImageReporter` and the `TemplateSrc` property to use your template.

```
import mlreportgen.report.*
rpt = Report();
tp = TitlePage();
tp.Image = 'myImage.jpg';
tp.Image = getImageReporter(tp);
tp.Image.TemplateSrc = 'MyTitlePageTemplate';
add(rpt, tp);
```

## **See Also**

`mlreportgen.report.TitlePage` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Report`

**Introduced in R2017b**

## getPubDateReporter

**Class:** `mlreportgen.report.TitlePage`

**Package:** `mlreportgen.report`

Get title page publication date reporter

### Syntax

```
reporter = getPubDateReporter(tp)
```

### Description

`reporter = getPubDateReporter(tp)` returns a reporter that the `TitlePage` reporter (`tp`) uses to format the value specified by the `PubDate` property. You use `getPubDateReporter` to customize the publication date alignment, position, and appearance of the publication date.

.

### Input Arguments

**tp — Title page of the report**

reporter object

Title page of the report, specified as a title page reporter object. To create the `tp` input, you use `mlreportgen.report.TitlePage`.

### Output Arguments

**reporter — Title page publication date reporter**

reporter object

Title page publication date reporter, returned as a reporter object.

### Examples

#### Use Nondefault Title Page Publication Date Style

Create a style for the publication date on your title page that differs from the default. Before you run this example, create a template file named `MyTitlePageTemplate` and customize its `TitlePagePubDate` style. Then, use `getPubDateReporter` and the `TemplateSrc` property to use your template.

```
import mlreportgen.report.*
rpt = Report();
tp = TitlePage();
tp.PubDate = 'June 1 2017';
tp.PubDate = getPubDateReporter(tp);
tp.PubDate.TemplateSrc = 'MyTitlePageTemplate';
add(rpt, tp);
```

## **See Also**

`mlreportgen.report.TitlePage` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Report`

**Introduced in R2017b**

## getPublisherReporter

**Class:** mlreportgen.report.TitlePage

**Package:** mlreportgen.report

Get title page publisher reporter

### Syntax

```
reporter = getPublisherReporter(tp)
```

### Description

`reporter = getPublisherReporter(tp)` returns a reporter that the TitlePage reporter (`tp`) uses to format the value specified by its `Publisher` property. You use `getPublisherReporter` to customize the publisher alignment, position, and appearance.

### Input Arguments

**tp — Title page of the report**

reporter object

Title page of the report, specified as a title page reporter object. To create the `tp` input, you use `mlreportgen.report.TitlePage`.

### Output Arguments

**reporter — Title page publisher reporter**

reporter object

Title page publisher reporter, returned as a reporter object.

### Examples

#### Use Nondefault Title Page Publisher Style

Create a style for the publisher on your title page that differs from the default. Before you run this example, create a template file named `MyTitlePageTemplate` and customize its `TitlePagePublisher` style. Then, use `getPublisherReporter` and the `TemplateSrc` property to use your template.

```
import mlreportgen.report.*
rpt = Report();
tp = TitlePage();
tp.Publisher = 'Report Publishing Company';
tp.Publisher = getPublisherReporter(tp);
tp.Publisher.TemplateSrc = 'MyTitlePageTemplate';
add(rpt, tp);
```

## **See Also**

`mlreportgen.report.TitlePage` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Report`

**Introduced in R2017b**

## getSubtitleReporter

**Class:** `mlreportgen.report.TitlePage`

**Package:** `mlreportgen.report`

Get title page subtitle reporter

### Syntax

```
reporter = getSubtitleReporter(tp)
```

### Description

`reporter = getSubtitleReporter(tp)` returns a reporter that the `TitlePage` reporter (`tp`) uses to format the value specified by its `Subtitle` property. You use `getSubtitleReporter` to customize the subtitle alignment, position, and appearance.

### Input Arguments

**tp — Title page of the report**

reporter object

Title page of the report, specified as a title page reporter object. To create the `tp` input, you use `mlreportgen.report.TitlePage`.

### Output Arguments

**reporter — Title page subtitle reporter**

reporter object

Title page subtitle reporter, returned as a reporter object.

### Examples

#### Use Nondefault Title Page Subtitle

Create a style for your title page subtitle that differs from the default. Before you run this example, create a template file named `MyTitlePageTemplate` and customize its `SubtitlePageTitle` style. Then, use `getSubtitleReporter` and the `TemplateSrc` property to use your template.

```
import mlreportgen.report.*
rpt = Report();
tp = TitlePage();
tp.Subtitle = 'My Report Subtitle';
tp.Subtitle = getSubtitleReporter(tp);
tp.Subtitle.TemplateSrc = 'MyTitlePageTemplate';
add(rpt, tp);
```



## **See Also**

`mlreportgen.report.TitlePage` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Report`

**Introduced in R2017b**

## getTitleReporter

**Class:** mlreportgen.report.TitlePage

**Package:** mlreportgen.report

Get title page title reporter

### Syntax

```
reporter = getTitleReporter(tp)
```

### Description

`reporter = getTitleReporter(tp)` returns a reporter that the TitlePage reporter (`tp`) uses to format the value specified by its `Title` property. Use `getTitleReporter` to customize the title alignment, position, and appearance.

### Input Arguments

**tp — Title page of the report**

reporter object

Title page of the report, specified as a title page reporter object. To create the `tp` input, you use `mlreportgen.report.TitlePage`.

### Output Arguments

**reporter — Title page title reporter**

reporter object

Title page title reporter, returned as a reporter object.

### Examples

#### Use Nondefault Title Page Title Style

Create a style for the title on your title page that differs from the default style. Before you run this example, create a template file named `MyTitlePageTemplate` and customize its `TitlePageTitle` style. Then, use the `getTitleReporter` method and the `TemplateSrc` property to use your template.

```
import mlreportgen.report.*
rpt = Report();
tp = TitlePage();
tp.Title = 'My Report Title'
tp.Title = getTitleReporter(tp);
tp.Title.TemplateSrc = 'MyTitlePageTemplate';
add(rpt, tp);
```

## **See Also**

`mlreportgen.report.TitlePage` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Report`

**Introduced in R2017b**

## mlreportgen.dom.getDefaultNumberFormat

**Package:** mlreportgen.dom

Get default formatting for numeric data generated by DOM API

### Syntax

```
formatSpec = mlreportgen.dom.getDefaultNumberFormat()
```

### Description

`formatSpec = mlreportgen.dom.getDefaultNumberFormat()` returns the default number format specification as a character vector. To set the default number format specification, use `mlreportgen.dom.setDefaultNumberFormat`.

### Examples

#### Get Default Number Format

Get the current default number format that the DOM API uses when representing a number as text.

```
import mlreportgen.dom.*  
  
rpt = Document("myReport", "pdf");  
  
numberFormat = getDefaultNumberFormat();  
p = Paragraph("Default number format is : ");  
p.append(numberFormat);  
  
append(rpt,p);  
  
close(rpt);  
rptview(rpt);
```

### Output Arguments

#### **formatSpec** — Format specification

character vector

Format specification, returned as a character vector. The specification is a valid format specification for the `sprintf` function and uses one of these operators:

- %f
- %e
- %E
- %g

- %G

If the format specification is an empty character array, a number is formatted using the maximum number of digits needed to represent the number accurately.

### **See Also**

`sprintf` | `mlreportgen.dom.setDefaultNumberFormat` | `mlreportgen.dom.Number` | `mlreportgen.dom.NumberFormat`

### **Topics**

“Format Numbers in a Table” on page 17-112

“Format Numbers” on page 13-43

**Introduced in R2021a**

# mlreportgen.dom.setDefaultNumberFormat

**Package:** mlreportgen.dom

Set default formatting of numeric data generated by DOM API

## Syntax

```
mlreportgen.dom.setDefaultNumberFormat(formatSpec)
```

## Description

`mlreportgen.dom.setDefaultNumberFormat(formatSpec)` sets the default number format specification for the DOM API. The default specification applies for the duration of the MATLAB session and applies to numbers in document elements, such as paragraphs, tables, and lists, and to numbers represented as `mlreportgen.dom.Number` objects. You can override the default specification for one number or all numbers in the document element by using an `mlreportgen.dom.NumberFormat` object.

## Examples

### Set Default Number Format

Set the default number format for the MATLAB session.

```
import mlreportgen.dom.*  
setDefaultNumberFormat("%0.2f");
```

Create a report and include a number with the default format in the report.

```
rpt = Document("Report with setDefaultNumberFormat", "pdf");  
  
n = Number(pi);  
append(rpt,n);  
  
close(rpt);  
rptview(rpt);
```

## Input Arguments

### formatSpec — Format specification

character vector | string scalar

Format specification, specified as a character vector or string scalar. The specification must be a valid format specification for the `sprintf` function and use one of these operators:

- %f
- %e

- %E
- %g
- %G

Example: "%0.2f"

## See Also

[sprintf](#) | [mlreportgen.dom.getDefaultNumberFormat](#) | [mlreportgen.dom.Number](#) | [mlreportgen.dom.NumberFormat](#)

## Topics

“Format Numbers in a Table” on page 17-112

“Format Numbers” on page 13-43

## Introduced in R2021a

## mlreportgen.utils.capitalizeFirstChar

**Package:** mlreportgen.utils

Capitalize first character of string

### Syntax

```
capOut = mlreportgen.utils.capitalizeFirstChar(strchar)
```

### Description

`capOut = mlreportgen.utils.capitalizeFirstChar(strchar)` capitalizes the first character of the input character vector or string scalar.

### Examples

#### Capitalize First Letter of Character Vector

```
strInput = 'this character vector needs an initial capital letter.';  
capOut = mlreportgen.utils.capitalizeFirstChar(strInput)
```

```
capOut  
    'This character vector needs an initial capital letter.'
```

### Input Arguments

#### **strchar** — Text whose first character is to be capitalized

character vector | string scalar

Text whose first character is to be capitalized, specified as a character vector or string scalar.

### Output Arguments

#### **capOut** — Text with first letter capitalized

character vector | string scalar

Text with first letter capitalized, specified as a character vector or string scalar. If the input is a character vector, the output is a character vector. If the input is a string scalar, the output is a string scalar.

### See Also

`mlreportgen.utils.makeSingleLineText` | `mlreportgen.utils.normalizeString`

**Introduced in R2018b**



# mlreportgen.utils.fileToURI

**Package:** mlreportgen.utils

Convert file path to Universal Resource Identifier (URI)

## Syntax

```
uri = mlreportgen.utils.fileToURI(filename)
```

## Description

`uri = mlreportgen.utils.fileToURI(filename)` converts `filename` to a Universal Resource Identifier (URI).

## Examples

### Convert File Name to URI

```
uri = mlreportgen.utils.fileToURI...  
    ("C:/Users/jsmith/Desktop/200-C Form.pdf")  
  
uri =  
  
    "file:///C:/Users/jsmith/Desktop/200-C%20Form.pdf"
```

## Input Arguments

### **filename** — Name of file to convert

character vector | string scalar

Name or full path of file to convert, specified as a character vector or string scalar.

## Output Arguments

### **uri** — Universal Resource Identifier version of file name

character vector | string scalar

Universal Resource Identifier (URI) of the file, returned as a character vector or string scalar. URIs are strings that identify resources. In particular, they specify the name of a file and the path to that file. By using a standard format, URIs allow access to a resource over a network. Web addresses (URLs) are types of URIs.

## See Also

`mlreportgen.utils.findFile`

**Introduced in R2018b**

## mlreportgen.utils.findFile

**Package:** mlreportgen.utils

Find file path

### Syntax

```
filepath = mlreportgen.utils.findFile(filename)
filepath = mlreportgen.utils.findFile(filename,Name=Value)
```

### Description

`filepath = mlreportgen.utils.findFile(filename)` returns the full file path from the specified file name. You can specify the file name with or without an extension.

`filepath = mlreportgen.utils.findFile(filename,Name=Value)` uses additional options specified by one or more Name-Value pair arguments.

### Examples

#### Find File Using Partial File Name

```
filepath = mlreportgen.utils.findFile("MyFile")

filepath =
    "C:\Users\username\Documents\MyFile"
```

#### Find File Using a File Extension List

```
filepath = mlreportgen.utils.findFile("HTMLFile",...
    FileExtensions=["docx" "rtf" "html"]);

filepath =
    "C:\Users\username\Documents\HTMLFile.html"
```

### Input Arguments

#### filename — File name

character vector | string scalar

File name with or without a file extension, specified as a character vector or string scalar.

#### Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

*Before R2021a, use commas to separate each name and value, and enclose Name in quotes.*

Example: `mlreportgen.utils.findFile("myFile",FileExtensions=["htm" "html"])`

### **FileExtensions – List of file extensions**

character vector | string array | cell array of character vectors

List of file extensions, specified as the comma-separated pair consisting of "FileExtensions" and a character vector, string array, or cell array of character vectors. `mlreportgen.utils.findFile` searches for a file name with one of the specified extensions or no extension.

Example: `mlreportgen.utils.findFile("myFile",FileExtensions=["docx" "rtf"])`

### **FileMustExist – Whether the file must exist**

true (default) | false

Specify whether the file to find must exist, specified as `true` or `false`. If `true`, the file must exist. If `false`, the file does not exist. In this case, only the full path of the file with the specified partial file name is created. No file with that path is created.

## **Output Arguments**

### **filepath – Path to file**

string scalar

Path to file, returned as a string scalar.

## **See Also**

`mlreportgen.utils.isFileLocked` | `mlreportgen.utils.fileToURI`

**Introduced in R2018b**

## mlreportgen.utils.hash

**Package:** mlreportgen.utils

Hash a piece of text

### Syntax

```
outstring = mlreportgen.utils.hash(intext)
```

### Description

`outstring = mlreportgen.utils.hash(intext)` generates an MD5 hash for a piece of text. An MD5 hash encodes the text into a 128-bit representation.

### Examples

#### Hash a Piece of Text

```
mlreportgen.utils.hash("This is a test string to hash.")
```

```
ans =
```

```
    "5d075889248b68d16b1fa9ad430fb4c8"
```

### Input Arguments

#### **intext** — Text to hash

character vector | string scalar

Text to hash, specified as a character vector or string scalar.

### Output Arguments

#### **outstring** — Hashed string

string scalar

Hashed representation of original text, returned as a string scalar.

### See Also

`mlreportgen.utils.normalizeString` | `mlreportgen.utils.toString`

**Introduced in R2018b**

# mlreportgen.utils.html2dom.prepHTMLFile

Prepare HTML file for conversion to DOM

## Syntax

```
preppedHTMLStr = mlreportgen.utils.html2dom.prepHTMLFile(htmlFile)
preppedHTMLFileName = mlreportgen.utils.html2dom.prepHTMLFile(htmlFile,
preppedHTMLFileName)
preppedHTMLStr = mlreportgen.utils.html2dom.prepHTMLFile( ___, "Tidy", false)
```

## Description

`preppedHTMLStr = mlreportgen.utils.html2dom.prepHTMLFile(htmlFile)` prepares the HTML in the file specified by `htmlFile` for conversion to the MATLAB Report Generator internal document object model (DOM). The prepared HTML in `preppedHTMLStr` can be converted to a DOM API representation by using an `mlreportgen.dom.HTML` object. The `mlreportgen.utils.html2dom.prepHTMLFile` function:

- Corrects invalid markup by calling `mlreportgen.utils.tidy` with the settings for HTML output.
- Uses the MATLAB web browser to convert the tidied markup to an HTML DOM document. See <https://www.w3.org/TR/WD-DOM/introduction.html>.

The MATLAB web browser computes the CSS properties of the elements in the HTML input based on internal and external style sheets specified by the input HTML, and on the style attribute of an element. The CSS property computation supports all valid CSS style sheet selectors, including selectors not directly supported by `mlreportgen.dom.HTML` or `mlreportgen.dom.HTMLFile` objects.

- Converts the HTML DOM document to HTML markup that is supported by `mlreportgen.dom.HTML` and `mlreportgen.dom.HTMLFile` objects. The style attribute for each element specifies the element CSS properties that the MATLAB web browser computed.
- Returns the prepared HTML as a string scalar.

`preppedHTMLFileName = mlreportgen.utils.html2dom.prepHTMLFile(htmlFile, preppedHTMLFileName)` generates the prepared HTML in a file with the name specified by `preppedHTMLFile`. The prepared HTML in `preppedHTMLFileName` can be converted to a DOM API representation by using an `mlreportgen.dom.HTMLFile` object.

`preppedHTMLStr = mlreportgen.utils.html2dom.prepHTMLFile( ___, "Tidy", false)` prepares the HTML without first tidying it. Specify "Tidy", false after all other input arguments. Use this syntax if you want to tidy the HTML markup yourself. For example, you might want to call `mlreportgen.utils.tidy` with different options than the ones used by `mlreportgen.utils.html2dom.prepHTMLFile`, then pass the tidied HTML as the input to `mlreportgen.utils.html2dom.prepHTMLFile`.

## Examples

### Prepare an HTML File for Conversion to a DOM Object

Use `mlreportgen.utils.html2dom.prepHTMLFile` to prepare an HTML file for conversion to a DOM object that you can append to a report.

Create a CSS style sheet, `myCSS.css`, to specify that the text in a paragraph is red and that the font family is Arial.

```
p {
    color: red;
    font-family: Arial;
}
```

Create a file, `myHTML.html`, that contains this HTML:

```
<html>
<head>
  <link rel="stylesheet" type="text/css" href="myCSS.css" >
</head>
<body>
  <p> Hello World</p>
</body>
</html>
```

The HTML is not XML-parsable because the `link` element is not properly closed. The slash `/` before the closing angle bracket `>` is missing.

Try to convert the HTML to a DOM object and append the object to a report.

```
import mlreportgen.dom.*;
rpt = Document("MyReport","docx");
htmlObj = HTMLFile("myHTML.html");
append(rpt,htmlObj);
close(rpt);
rptview(rpt);

Error using mlreportgen.dom.HTMLFile
HTML error: expected end of tag 'link'
```

`mlreportgen.dom.HTMLFile` ends with an error due to the missing end tag.

Prepare the HTML for conversion to DOM by using `mlreportgen.utils.html2dom.prepHTMLFile`. Create an `mlreportgen.dom.HTMLFile` object from the prepared HTML and append the object to the report.

```
import mlreportgen.dom.*
import mlreportgen.utils.html2dom.*
d = Document("test","pdf");
preppedHTMLFile = prepHTMLFile("myHTML.html","mypreppedHTML.html");
htmlObj = HTMLFile(preppedHTMLFile);
append(d,htmlObj);
```

```
close(d);  
rptview(d);
```

## Input Arguments

### **htmlFile** – HTML file

character vector | string scalar

HTML file to be prepared for conversion to DOM, specified as a character vector or string scalar.

### **preppedHTMLFileName** – File for prepared HTML

character vector | string scalar

File for prepared HTML, specified as a character vector or string scalar.

Example: "myHTML.html"

## Output Arguments

### **preppedHTMLStr** – Prepared HTML

string scalar

Prepared HTML, returned as a string scalar.

### **preppedHTMLFileName** – Name of file containing prepared HTML

string scalar

Name of the file that contains the prepared HTML, returned as a string scalar.

## Tips

- MATLAB Report Generator `mlreportgen.dom.HTML` or `mlreportgen.dom.HTMLFile` objects typically cannot accept the raw HTML output of third-party applications, such as Microsoft Word, that export native documents as HTML markup. In these cases, your Report API report generation program can use the `mlreportgen.utils.html2dom.prepHTMLString` and `mlreportgen.utils.html2dom.prepHTMLFile` functions to prepare the raw HTML for use with the `mlreportgen.dom.HTML` or `mlreportgen.dom.HTMLFile` objects. Typically, your program will have to further process the prepared HTML to remove valid but undesirable objects, such as line feeds that were in the raw content.

## See Also

`mlreportgen.utils.html2dom.prepHTMLString` | `mlreportgen.dom.HTML` |  
`mlreportgen.dom.HTMLFile` | `mlreportgen.utils.tidy`

## Topics

“Prepare HTML for Conversion to a DOM Object” on page 13-108

“Convert HTML Content to a DOM Object” on page 13-105

“Requirements for Converting HTML to a DOM Object” on page 13-110

## Introduced in R2020a

## mlreportgen.utils.html2dom.prepHTMLString

Prepare HTML string for conversion to DOM

### Syntax

```
preppedHTMLStr = mlreportgen.utils.html2dom.prepHTMLString(htmlStr)
preppedHTMLStr = mlreportgen.utils.html2dom.prepHTMLString(
htmlStr, "Tidy", false)
```

### Description

`preppedHTMLStr = mlreportgen.utils.html2dom.prepHTMLString(htmlStr)` prepares the HTML in the string specified by `htmlStr` for conversion to the MATLAB Report Generator internal document object model (DOM). The prepared HTML in `preppedHTMLStr` can be converted to a DOM API representation by using an `mlreportgen.dom.HTML` object. The `mlreportgen.utils.html2dom.prepHTMLString` function:

- Corrects invalid markup by calling `mlreportgen.utils.tidy` with the settings for HTML output.
- Uses the MATLAB web browser to convert the tidied markup to an HTML DOM document. See <https://www.w3.org/TR/WD-DOM/introduction.html>.

The MATLAB web browser computes the CSS properties of the elements in the HTML input based on internal and external style sheets specified by the input HTML, and on the style attribute of an element. The CSS property computation supports all valid CSS style sheet selectors, including selectors not directly supported by `mlreportgen.dom.HTML` objects.

- Converts the HTML DOM document to HTML markup that is supported by `mlreportgen.dom.HTML` objects. The style attribute for each element specifies the element CSS properties that the MATLAB web browser computed.
- Returns the prepared HTML as a string scalar.

`preppedHTMLStr = mlreportgen.utils.html2dom.prepHTMLString(htmlStr, "Tidy", false)` prepares the input HTML without first tidying it. Use this syntax if you want to tidy the HTML markup yourself. For example, you might want to call `mlreportgen.utils.tidy` with different options than the ones used by `mlreportgen.utils.html2dom.prepHTMLString`, then pass the tidied HTML as the input to `mlreportgen.utils.html2dom.prepHTMLString`.

### Examples

#### Prepare HTML for Conversion to a DOM Object

Use `mlreportgen.utils.html2dom.prepHTML` to prepare an HTML string for conversion to a DOM object that you can append to a report.

Create a string named `myHTMLStr` that has this HTML content:

```
myHTMLStr = "<html><body><p>This is the first paragraph.<p>This is the second paragraph.<p></body></html>";
```



The paragraph is missing the / in the end tag </p>.

Try to convert the HTML to a DOM object and append the object to a report.

```
import mlreportgen.dom.*;
rpt = Document("MyReport","docx");
htmlObj = mlreportgen.dom.HTML(myHTMLStr);
append(rpt,htmlObj);
close(rpt);
rptview(rpt);
```

```
Error using mlreportgen.dom.HTML
HTML error: expected end of tag 'p'
```

mlreportgen.dom.HTML ends with an error due to the missing end tag.

Prepare the HTML by using mlreportgen.utils.html2dom.prepHTMLString. Create an mlreportgen.dom.HTML object from the prepared HTML and append the object to the report.

```
import mlreportgen.dom.*;
rpt = Document("MyReport","docx");
%prepare the HTML
myPreppedHTML = mlreportgen.utils.html2dom.prepHTMLString(myHTMLStr);
htmlObj = mlreportgen.dom.HTML(myPreppedHTML);
append(rpt,htmlObj);
close(rpt);
rptview(rpt);
```

## Input Arguments

### htmlStr — HTML content

character vector | string scalar

HTML content to be prepared for conversion to the DOM, specified as a character vector or string scalar.

## Output Arguments

### preppedHTMLStr — Prepared HTML

string scalar

Prepared HTML, returned as a string scalar.

## Tips

- MATLAB Report Generator mlreportgen.dom.HTML or mlreportgen.dom.HTMLFile objects typically cannot accept the raw HTML output of third-party applications, such as Microsoft Word, that export native documents as HTML markup. In these cases, your Report API report generation program can use the mlreportgen.utils.html2dom.prepHTMLString and mlreportgen.utils.html2dom.prepHTMLFile functions to prepare the raw HTML for use with the mlreportgen.dom.HTML or mlreportgen.dom.HTMLFile objects. Typically, your program will have to further process the prepared HTML to remove valid but undesirable objects, such as line feeds that were in the raw content.

**See Also**

`mlreportgen.dom.HTML` | `mlreportgen.dom.HTMLFile` |  
`mlreportgen.utils.html2dom.prepHTMLFile` | `mlreportgen.utils.tidy`

**Topics**

“Convert HTML Content to a DOM Object” on page 13-105  
“Prepare HTML for Conversion to a DOM Object” on page 13-108  
“Requirements for Converting HTML to a DOM Object” on page 13-110

**Introduced in R2020a**

# mlreportgen.utils.isFileLocked

**Package:** mlreportgen.utils

Determine if file is locked

## Syntax

```
tf = mlreportgen.utils.isFileLocked(filename)
```

## Description

`tf = mlreportgen.utils.isFileLocked(filename)` tests whether the specified file is locked.

## Examples

### Check for Locked File

Verify that the `output.pdf` file is locked.

```
mlreportgen.utils.isFileLocked('output.pdf')
```

```
ans =  
    logical  
    0
```

## Input Arguments

**filename** — Name of file to test

character vector | string scalar

Name or full path of the file to test, specified as a character vector or string scalar.

## Output Arguments

**tf** — Status of the file

true | false

Status of the file, returned as `true` or `false`. If the file is locked by another process, `true` is returned. If the file is not locked, `false` is returned.

## See Also

`mlreportgen.utils.findFile`

**Introduced in R2018b**

## mlreportgen.utils.makeSingleLineText

**Package:** mlreportgen.utils

Convert input to single line

### Syntax

```
singleline = mlreportgen.utils.makeSingleLineText(strchar_in)
singleline = mlreportgen.utils.makeSingleLineText(strchar_in,delim)
```

### Description

`singleline = mlreportgen.utils.makeSingleLineText(strchar_in)` converts the input to a single line of text. Single spaces replace the line feeds and carriage returns in the input.

`singleline = mlreportgen.utils.makeSingleLineText(strchar_in,delim)` specifies the delimiter to use to replace line feeds and carriage returns.

### Examples

#### Convert Character Array to Single Line

```
devTitle = ['Thomas R. Lee'; ...
            'Sr. Developer'; ...
            'SFTware Corp.'];
singleline = mlreportgen.utils.makeSingleLineText(devTitle)

singleline =

    'Thomas R. Lee Sr. Developer SFTware Corp.'
```

### Input Arguments

#### strchar\_in — Input to convert

character vector | string scalar | cell array of character vectors | string array | numeric array

Input to convert to single line of text, specified as a character array, string array, cell array of characters, or numeric array.

#### delim — Delimiter

character vector | string scalar

Delimiter to substitute for each line feed and carriage return, specified as a character vector or string scalar.

### Output Arguments

#### singleline — Single line of text

character vector | string scalar

Single line of text, returned as a character vector or string scalar. The formatting of the output depends on the input type.

<b>Input</b>	<b>Output</b>
character vector or numeric array	Character vector with line feeds and carriage returns removed.
string scalar	String scalar with line feeds and carriage returns removed.
cell array of character vectors	Character vector with line feeds and carriage returns removed. Entries in the array are concatenated and separated using the delimiter.
string array	String scalar with line feeds and carriage returns removed. Entries in the array are concatenated and separated using the delimiter.

### **See Also**

`mlreportgen.utils.capitalizeFirstChar` | `mlreportgen.utils.hash` |  
`mlreportgen.utils.normalizeString` | `mlreportgen.utils.toString`

**Introduced in R2018b**

## mlreportgen.utils.normalizeLinkID

Normalize a link target ID

### Syntax

```
outID = mlreportgen.utils.normalizeLinkID(inID)
```

### Description

`outID = mlreportgen.utils.normalizeLinkID(inID)` converts the input link target ID to an ID that is valid for Microsoft Word, PDF, and HTML reports. The output ID consists of the prefix `mw_` and an MD5 hash of the input ID. The generated ID conforms to the Word limitation on ID length and the PDF requirement that an ID begin with an alphabetic character.

### Examples

#### Create a Link Target with Normalized ID

This example generates a report where one paragraph links to another paragraph. The link target ID of the target paragraph is generated by using `mlreportgen.utils.normalizeLinkID`.

```
import mlreportgen.dom.*
import mlreportgen.utils.*
d = Document('mydoc', 'pdf');

p = Paragraph('This is my paragraph');
linkID = normalizeLinkID('home');
append(p, LinkTarget(linkID));
append(d, p);

p = Paragraph('This is another paragraph');
p.Style = {PageBreakBefore(true)};
append(d, p);
append(d, InternalLink(linkID, 'Go to Top'));

close(d);
rptview(d);
```

### Input Arguments

#### **inID** — Link target ID to be normalized

string scalar | character vector

Link target ID to be normalized, specified as a string scalar or character vector.

### Output Arguments

#### **outID** — Normalized link target ID

string scalar

Normalized link target ID, returned as a string scalar.

## Tips

To generate a link target ID for a Simulink or Stateflow object, use `mlreportgen.utils.getObjectID`. This function generates a link target ID that is valid for all report types.

## See Also

`mlreportgen.dom.LinkTarget` | `mlreportgen.dom.InternalLink`

## Topics

“Create Links” on page 13-82

**Introduced in R2020a**

## mlreportgen.utils.normalizeString

**Package:** mlreportgen.utils

Remove extra spaces and line feeds from text

### Syntax

```
n_strchar = mlreportgen.utils.normalizeString(strchar)
```

### Description

`n_strchar = mlreportgen.utils.normalizeString(strchar)` normalizes text by removing leading and trailing spaces and replacing carriage returns and tabs with a single space. The returned value has the same type as the input type.

### Examples

#### Normalize String Arrays and Character Vectors

```
import mlreportgen.utils.*
str = "  a sample string "
n_strchar = normalizeString(str)

str =
    "  a sample string "

n_strchar =
    "a sample string"

import mlreportgen.utils.*
char_vec = ' a sample character vector'
char_vec1 = [char_vec newline ' plus a new line ']
n_strchar = normalizeString(char_vec1)

char_vec =
    ' a sample character vector '

char_vec1 =
    ' a sample character vector
    plus a new line '

n_strchar =
    'a sample character vector plus a new line'
```

### Input Arguments

#### **strchar** — Text to normalize

character vector | string array

Text to normalize, specified as a character vector or string array.



## Output Arguments

### **n\_strchar** — Normalized text

character vector | string array

Normalized text, returned as a character vector or string array, depending on the input type.

### **See Also**

`mlreportgen.utils.toString` | `mlreportgen.utils.capitalizeFirstChar`

**Introduced in R2018b**

## mlreportgen.utils.tidy

**Package:** mlreportgen.utils

Correct and clean XML and HTML content

### Syntax

```
outString = mlreportgen.utils.tidy(inString)
outFile = mlreportgen.utils.tidy(inFile)
___ = mlreportgen.utils.tidy( ____,Name=Value)
```

### Description

`outString = mlreportgen.utils.tidy(inString)` corrects and cleans an XHTML string. Correcting adds missing end tags. Cleaning removes unnecessary tags.

`outFile = mlreportgen.utils.tidy(inFile)` corrects and cleans an XHTML file.

`___ = mlreportgen.utils.tidy( ____,Name=Value)` uses additional options specified by one or more Name-Value pair arguments.

### Examples

#### Tidy an Input String

```
outString = mlreportgen.utils.tidy...
    ("<p>sample input string with missing end tag")
```

```
outString =
```

```
"<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
  <title></title>
  </head>
  <body>
  <p>sample input string with missing end tag</p>
  </body>
</html>
"
```

#### Tidy and Overwrite Input File

For this example, substitute your username in the "c:\Users\username\Documents\myHTMLFile.html" string.

```
outFile = mlreportgen.utils.tidy("myHTMLFile.html",...
    OutputFile="C:\Users\username\Documents\myHTMLFile.html");
```

```
outFile =
    "C:\Users\username\Documents\myHTMLFile.html"
```

### Write to New File and Use Custom Configuration File

Assume that you have created your own configuration file, named `myConfig.cfg`, and stored it in your Documents folder while you test it. For ease of finding the file later, store it in the same location as the default configuration files or store it with your output file.

For this example, substitute your username in the `"c:\Users\username\Documents\myNewHTMLFile.html"` string.

```
outFile = mlreportgen.utils.tidy("myHTMLFile.html",...
    OutputFile="c:\Users\username\Documents\myNewHTMLFile.html",...
    ConfigFile="myConfig.cfg");

outFile =
    "C:\Users\username\Documents\myHTMLFile.html"
```

## Input Arguments

### **inString** — HTML text to correct and clean

string | character vector

HTML text to correct and clean, specified as a string or character vector.

### **inFile** — HTML file to correct and clean

string | character vector

HTML file to correct and clean, specified as a string or character vector.

### Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

*Before R2021a, use commas to separate each name and value, and enclose Name in quotes.*

Example: `tidy("myFile.html",OutputType="html")`

### **OutputType** — Type of output file

string | character vector

Type of output file, specified as a string or character vector. Valid values are `'xml'`, `'html'`, and `'xhtml'`. To ensure that the tidied file is XML compliant, use `'xhtml'` as the output type.

### **OutputFile** — Path of output file

string | character vector

Path of output file, specified as a string or character vector. If the file is in the current working folder, you can specify only the file name, otherwise specify the full path. The tidied output file can overwrite the original HTML file or be saved to a new file.

**ConfigFile — Configuration file**

string | character vector

Configuration file, specified as a string or character vector. The configuration file contains options for cleaning and correcting input strings and files. (For more information, see External Links.) Default configuration files for each output type are located in the `<matlabroot>/toolbox/shared/mlreportgen/utils/resources` folder. The files are `tidy-html.cfg`, `tidy-xml.cfg`, and `tidy-xhtml.cfg`. You can create your own configuration file and specify it using this parameter. The easiest way to create your own configuration file is to copy the default file, make your changes, and save it using a new file name. If you specify your own `ConfigFile`, it overrides the `OutputType` parameter.

---

**Note** The indentation of the tidied file is set to `false` in the default configuration file. To turn on indentation, create your own configuration file and set `indent` to `true`.

---

## Output Arguments

**outString — Tidied XHTML string**

string

Tidied XHTML, returned as a string that contains the basic elements of an HTML file.

**outFile — Tidied XHTML file**

string

Tidied XHTML file, returned as a string that indicates the file location and name.

## See Also

`mlreportgen.utils.HTMXDoc` | `mlreportgen.utils.HTMLDoc`

### External Websites

HTML Tidy Options Quick Reference

**Introduced in R2018b**

# mlreportgen.utils.toString

**Package:** mlreportgen.utils

Create string representation of MATLAB variable

## Syntax

```
convValue = mlreportgen.utils.toString(varName)
```

## Description

`convValue = mlreportgen.utils.toString(varName)` converts the value of a MATLAB variable to a string.

## Examples

### Convert MATLAB Variable Values to String

```
num = 10;
convValue = mlreportgen.utils.toString(num)

convValue =
    "10"

matrix = [1,7,10;3,1,6];
convValue = mlreportgen.utils.toString(matrix)

convValue =
    "[1  7  10 ;
     3  1  6 ]"
```

## Input Arguments

**varName** — MATLAB variable

variable name

MATLAB variable whose value is to be converted to a string.

## Output Arguments

**convValue** — Converted variable value

string scalar

Converted MATLAB variable value, returned as a string scalar

## See Also

[mlreportgen.utils.makeSingleLineText](#) | [mlreportgen.utils.normalizeString](#) | [mlreportgen.utils.capitalizeFirstChar](#)

**Introduced in R2018b**

# pptview

Open Microsoft PowerPoint presentation or convert it to PDF

## Syntax

```
[Status,Message] = pptview(filename)
```

```
[Status,Message] = pptview(filename,conversionMode)
```

```
[Status,Message] = pptview(filename,'closeapp','closedoc')
```

```
[Status,Message] = pptview(filename,'closedoc')
```

## Description

[Status,Message] = pptview(filename) opens the specified presentation file. The application used to open the file depends on the platform:

- Windows — pptview opens the file in PowerPoint.
- Linux — pptview calls the soffice command to open the file in Apache OpenOffice or LibreOffice, depending on which application is installed.
- Macintosh — pptview calls the soffice command to open the file in Apache OpenOffice, which must be installed in the /Applications folder.

[Status,Message] = pptview(filename,conversionMode) converts the specified PowerPoint file to PDF on PCs with Microsoft Office installed.

[Status,Message] = pptview(filename,'closeapp','closedoc') closes the PPT application and presentation file.

[Status,Message] = pptview(filename,'closedoc') closes the presentation file.

## Examples

### Open a PPT File

Open a presentation.

```
pptview('myppt');
```

### Convert a PPT File to PDF Format

Convert a PPT file to PDF format and close the PPT file. Run this command only on PCs with Microsoft Office installed.

```
pptview('myppt','converttopdf');
```

## Input Arguments

### **filename** — PPT file to open or convert

character vector

PPT file to open or convert, specified as a character vector of a file name on the MATLAB path or the full path name. You do not need to include the file extension.

Example: 'myppt', 'MyPresentations/myppt'

### **conversionMode** — Method of converting to PDF

'converttopdf' | 'showaspdf'

Method of converting the PPT file to PDF, specified as one of these values:

- 'converttopdf' — Convert to PDF without displaying the results.
- 'showaspdf' — Convert to PDF and display the results in the PDF Viewer.

## Output Arguments

### **Status** — Success indicator

0 | 1

Success indicator for opening or converting the PPT file, returned as 1 for success and 0 for failure.

### **Message** — Error or warning information

character vector

Error or warning information about opening or converting the file, returned as a character vector.

## See Also

rptview | rptconvert

**Introduced in R2016a**



# report

Generate report from setup file

## Syntax

```
report
report (setup1,...,setupN)
[report1,report2,...,reportN] = report(setup1,setup2,...,setupN)
report(systemname)
report ( ____, -fFORMAT)
report ( ____, -fFORMAT, -sTEMPLATE)
report ( ____, -fFORMAT, -sSTYLESHEET)
report ( ____, -oPATH)
report ( ____, -genOption)
```

## Description

- `report` with no arguments opens the Report Explorer. For more information on the Report Explorer, see “Working with the Report Explorer” on page 1-16
- `report (setup1,...,setupN)` generates reports from the specified report setup files. The file names are character arrays that specify the paths of the setup files. You can specify one or more report setup files.
- `[report1,report2,...,reportN] = report(setup1,setup2,...,setupN)` returns the names of the generated reports. If an error occurs when generating a report, the returned `reportN` name is empty.
- `report(systemname)` generates a report specified by the `ReportName` property of a Simulink system or its parent. If the system does not specify a report name, this syntax runs the report specified by the parent system's `ReportName` property.
- `report ( ____, -fFORMAT)` specifies the output format of the report. To display a list of format IDs for template-based reports, use `rptconvert(' -domformatlist')`. To display a list of legacy format IDs, use `rptconvert(' -formatlist')`. You can use the template-based formats to convert both template-based reports and legacy reports. You can use the legacy formats only for converting XML files that use XSL or DSSSL style sheets to specify formatting. For example, to specify PDF output for a template-based report, use `-fdom-pdf`.
- `report ( ____, -fFORMAT, -sTEMPLATE)` specifies the id of the template of type `FORMAT` (or style sheet, for legacy reports) to be used to generate a report. To display a list of templates available to format template-based documents, use `rptconvert(' -templatelist',FORMAT)`. You should specify a template that is compatible with the specified document format. For example, use a PDF template for a converted PDF document.
- `report ( ____, -fFORMAT, -sSTYLESHEET)` specifies the id of the style sheet to be used to generate a report. To display a list of style sheets available to format documents of type `FORMAT`, use `rptconvert(' -stylesheet',FORMAT)`. Use a style sheet only for XML documents that specify a style sheet and only if `FORMAT` specifies a legacy output type, such as Adobe Acrobat.
- `report ( ____, -oPATH)` sets the name of the generated report. You can specify a path or a single file name for the `PATH`. If you specify a single file name, the report is generated in the directory specified by the report setup file.

- `report ( ____, -genOption)` specifies one of the following report generation options. You can specify more than one option.
  - `-noview` — Generates, but does not display, a report
  - `-graphical` — Opens the report in the Report Explorer and runs the report
  - `-debug` — Generates report debugging info at the MATLAB command line
  - `-quiet` — Suppresses command-line output during report generation

## Examples

### Set Generated Report Format

- Generate the report `testrpt` in PDF format:  

```
report testrpt -fpdf
```
- Generate the report `testrpt` in RTF format:  

```
report testrpt -frtf
```
- Generate the report `testrpt` in Microsoft Word format:  

```
report testrpt -fdoc
```

---

**Note** Only Microsoft Windows platforms support this option.

---

- Generate a multipage HTML report from the `figloop-tutorial` report setup file:  

```
report figloop-tutorial -fhtml -shtml-!MultiPage
```

### Generate a PDF Report Using the Default PDF Template

Generate a PDF report named `magic-square` using the default PDF template:

```
report('magic-square', '-fdom-pdf-direct', '-sdefault-rg-pdf')
```

### Generate a PDF Report from a Simulink System

Generate a default report in PDF format on the Simulink model `vdp`. To generate this report, you need Simulink installed.

```
report vdp -fPDF
```

### Specify File and Path of Generated Report

Generate a report named `simple-report` in the folder `/tmp/index.html`:

```
report ('simple-report', '-o/tmp/index.html')
```

### Generate a Report in the Directory Specified in the Report Setup File

Generate a report in the directory specified by `simple-report.rpt`:

```
report('simple-report', '-oindex.html')
```

## **See Also**

setedit | rptconvert | rptlist | compwiz

## **Topics**

“Generate Reports”

## rptconvert

Convert DocBook XML files into supported document formats

### Syntax

```
rptconvert()  
rptname = rptconvert (source)  
rptname = rptconvert (source, format)  
rptname = rptconvert (source, format, stylesheet)  
...=rptconvert(..., '-view')  
...=rptconvert(..., '-quiet')  
...=rptconvert(..., '-verbose')  
sheetlist = rptconvert('-stylesheetlist')  
sheetlist = rptconvert('-stylesheetlist', format)  
FORMATLIST = rptconvert('-formatlist')
```

### Description

This function converts a DocBook XML source file created by the report-generation process to a supported document format.

`rptconvert()` with no input arguments launches the converter. When input arguments are passed to this function, `rptconvert` converts the XML document to the specified format and displays status messages to the MATLAB Command Window.

`rptname = rptconvert (source)` converts the specified DocBook XML file created by the report-generation process. You can specify this file name with or without its file extension.

`rptname = rptconvert (source, format)` converts using the specified unique identifier code for each output format type. If you omit this argument, the XML file is converted to HTML format by default.

`rptname = rptconvert (source, format, stylesheet)` converts using the specified style sheet. If you omit this argument, the default style sheet for the selected `format` is used.

You can also pass the following flags to the input arguments:

- `...=rptconvert(..., '-view')` displays the converted document.
- `...=rptconvert(..., '-quiet')` suppresses status messages.
- `...=rptconvert(..., '-verbose')` shows detailed status messages.
- `sheetlist = rptconvert('-stylesheetlist')` returns a two-column cell array. The first column of this array includes valid style sheet identifiers. The second column includes descriptions of each style sheet.
- `sheetlist = rptconvert('-stylesheetlist', format)` returns an array like that returned by `sheetlist = rptconvert('-stylesheetlist')`. The first column of this array includes style sheet identifiers for the specified `format`.
- `FORMATLIST = rptconvert('-formatlist')` returns a two-column cell array. The first column of this array includes valid `format` values, the second column includes descriptions of each format.

## Examples

Retrieve a list of available HTML style sheets:

```
rptconvert('-stylesheetlist','html')
```

## See Also

setedit | report | rptlist | compwiz

## Topics

“Convert XML Documents to Different File Formats” on page 5-13

## **rptlist**

Retrieve list of all report setup files in MATLAB path

### **Syntax**

```
rptlist  
rptlist ('system_name')  
list = rptlist
```

### **Description**

`rptlist` with no arguments opens the Report Explorer, which lists available report setup files in the MATLAB path.

`rptlist ('system_name')` opens the Report Explorer with the Simulink system's `ReportName` property selected.

`list = rptlist` returns a list of report setup files in the MATLAB path.

### **See Also**

`setedit`

# rptrebuildcache

Rebuild Report Explorer template cache

## Syntax

```
rptrebuildcache
```

## Description

`rptrebuildcache` rebuilds the template cache of the Report Explorer. Run this command if you add a template to the MATLAB path after opening the Report Explorer. Another instance of when to run this command is if you add a template after you run a report setup file for the first time in a MATLAB session.

---

**Note** A template must reside on the MATLAB path for this command to cache it.

---

The first time you open the Report Explorer in a MATLAB session, the Report Explorer searches the MATLAB path for templates. It also searches the path the first time you run a report setup file. The Report Explorer enters the templates it finds in a cache. The report setup file specifies the template to use for the report. If the template is not in the current folder and you run a report setup file, the Report Explorer looks for the template in the cache. If you then create a template and do not run this command, the template is found only if it is in the current folder. To run the report from another folder, run this command after adding the template.

## See Also

`setedit` | `rptconvert` | `report` | `rptrebuildregistry`

**Introduced in R2017b**

## rptrebuildregistry

Rebuild Report Explorer style sheet registry

### Syntax

```
rptrebuildregistry
```

### Description

`rptrebuildregistry` rebuilds the XSL and DSSSL style sheet registry of the Report Explorer. Run this command if you add a style sheet to the MATLAB path after opening the Report Explorer. Another instance of when to run this command is if you add a style sheet after you run a report setup file for the first time in a MATLAB session.

---

**Note** A style sheet must reside on the MATLAB path for this command to register it.

---

The first time you open the Report Explorer in a MATLAB session, the Report Explorer searches the MATLAB path for XSL and DSSSL style sheets. It also searches the path the first time you run a report setup file. The Report Explorer enters those style sheets in a registry. The report setup file specifies the style sheet to use for the report. If the style sheet is not in the current folder and you run a report setup file, the Report Explorer looks for the style sheet in the registry. If you then create a style sheet and do not run this command, the style sheet is found only if it is in the current folder. To run the report from another folder, run this command after adding the style sheet.

### See Also

`setedit` | `rptconvert` | `report` | `rptrebuildcache`

**Introduced in R2017b**



# rptview

Display report or presentation

## Syntax

```
rptview(domObj)
rptview(reportPath)
rptview(reportName, format)
```

```
rptview(docxfile, 'pdf')
```

```
rptview(pptObj)
rptview(pptPath)
```

## Description

`rptview(domObj)` displays the report specified by the input `mlreportgen.dom.Document` object in an appropriate viewer.

To open a Word document on a Linux or Macintosh platform, `rptview` calls the `soffice` command. On Linux, Apache OpenOffice or LibreOffice must be installed. On Macintosh, Apache OpenOffice must be installed in the `/Applications` folder.

---

**Note** The `mlreportgen.report.Report` class has a method named `rptview`. When you call `rptview` with an `mlreportgen.report.Report` object as the argument value, the method is invoked. For all other types of argument values, the `rptview` function is invoked. See “Tips” on page 11-389.

---

`rptview(reportPath)` displays the report specified by `reportPath` in an appropriate viewer, based on the file extension.

`rptview(reportName, format)` displays the report specified by `reportName` in an appropriate viewer, based on the format specified in `format`.

`rptview(docxfile, 'pdf')` converts a Microsoft Word report to PDF and displays the report in a PDF viewer.

`rptview(pptObj)` displays the presentation specified by the input `mlreportgen.ppt.Presentation` object in Microsoft PowerPoint.

To open a presentation on a Linux or Macintosh platform, `rptview` calls the `soffice` command. On Linux, Apache OpenOffice or LibreOffice must be installed. On Macintosh, Apache OpenOffice must be installed in the `/Applications` folder.

`rptview(pptPath)` displays the presentation located at `pptPath` in Microsoft PowerPoint.

## Examples

### Display DOM Document in HTML Viewer

Display an HTML report that was generated using an `mlreportgen.dom.Document` object.

```
import mlreportgen.dom.*;
d = Document('mydoc');

p = Paragraph('Hello World');
append(d,p);

close(d);
rptview('mydoc');
```

### Display DOM Document in PDF Viewer

Display a PDF report that was generated using an `mlreportgen.dom.Document` object.

```
import mlreportgen.dom.*;
d = Document('mydoc','pdf');

append(d, 'Hello World');

close(d);
rptview(d);
```

### Display Report Using the OutputPath Property

Display a report that was generated using an `mlreportgen.report.Report` object. Specify the path and file name of the report by using the value of the `OutputPath` property of the object.

```
import mlreportgen.dom.*;
import mlreportgen.report.*;
rpt = Report('myReport','docx');

p = Paragraph('Hello World');
add(rpt,p);
close(rpt);
rptview(rpt.OutputPath);
```

### Convert Word Report to PDF and Display It

Use the `rptview` function to convert a Word report to PDF and display it in a PDF viewer.

```
import mlreportgen.dom.*;
import mlreportgen.report.*;
rpt = Report('myReport','docx');

p = Paragraph('Hello World');
add(rpt,p);
```

```
close(rpt);
rptview('myReport.docx', 'pdf');
```

### Display Word Report Based on Name

Create two reports with the same name, but with different formats and content. Specify the format to display the appropriate report.

```
import mlreportgen.dom.*;
import mlreportgen.report.*
rpt = Report("myReport", "html");

p = Paragraph("Hello World");
add(rpt,p);
close(rpt);

rptWord = Report("myReport", "docx");
p = Paragraph("Hello again, World");
add(rptWord,p);
close(rptWord);

rptview("myReport", "docx");
```

### Display Presentation Based on Presentation Object

Display a presentation by calling `rptview` with the name of the `mlreportgen.ppt.Presentation` object.

```
import mlreportgen.ppt.*
ppt = Presentation("MyPresentation");
open(ppt);

slidel = add(ppt, "Title and Table");
replace(slidel, "Title", "Magic Square Slide 1");
replace(slidel, "Table", Table(magic(3)));
close(ppt);
rptview(ppt);
```

### Display Presentation Based on Path and File Name

Display a presentation by calling `rptview` with the path and file name, including the extension, of the generated PowerPoint presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("MyPresentation");
open(ppt);

slidel = add(ppt, "Title and Table");
replace(slidel, "Title", "Magic Square Slide 1");
replace(slidel, "Table", Table(magic(3)));
```

```
close(ppt);
rptview("MyPresentation.pptx");
```

## Input Arguments

### **domObj** — Document object that generates report to view

`mlreportgen.dom.Document` object

Document object that generates the report to view, specified as an `mlreportgen.dom.Document` object.

### **reportPath** — Report file path and name

string scalar | character vector

Path and name of a specific report file, including the file extension, specified as a string scalar or character vector. You can use the `OutputPath` property of an `mlreportgen.dom.Document` object or an `mlreportgen.report.Report` object to provide `reportPath`.

The report file name extension determines the viewer in which the report displays.

File Extension	Viewer
.htm	MATLAB web browser
.html	MATLAB web browser
.zip	MATLAB web browser
.docx	Microsoft Word
.pdf	PDF viewer

### **reportName** — Report name

string scalar | character vector

Path and file name of a report, without the file extension, specified as a string scalar or character vector.

### **format** — Report output format

'html' | 'html-file' | 'docx' | 'pdf'

Report output format, specified as one of these values:

- 'html'
- 'html-file'
- 'docx'
- 'pdf'

### **docxfile** — Word .docx file to convert to PDF

string scalar | character vector

Word .docx file to convert to PDF, specified as a Word file having a .docx extension.

### **pptObj** — Presentation object that generates the presentation

`mlreportgen.ppt.Presentation`

Presentation object that generates the presentation to view, specified as an `mlreportgen.ppt.Presentation` object.

### **pptPath — Presentation file path and name**

string scalar | character vector

Path and name of a specific presentation file, including the file extension, specified as a string scalar or character vector. The file extension can be `.pptx` or `.potx`. You can use the `OutputPath` property of the `mlreportgen.ppt.Presentation` object to provide `pptPath`.

## **Tips**

Calling `rptview` with an `mlreportgen.report.Report` object as the argument value, invokes the `rptview` method of the `mlreportgen.report.Report` object. The `rptview` method calls the `rptview` function with the value of the `Document` property of the report object as the argument value.

Calling `rptview` with an argument value other than an `mlreportgen.report.Report` object invokes the `rptview` function.

For example, in the following code, the first `rptview` call invokes the `rptview` method. The second `rptview` call invokes the `rptview` function.

```
import mlreportgen.report.*
rpt = Report('myReport', 'pdf');
add(rpt, 'Hello World');
close(rpt);
% Invokes rptview method
rptview(rpt);
% Invokes rptview function
rptview('myReport.pdf');
```

## **See Also**

`mlreportgen.dom.Document` | `docview` | `rptview` | `mlreportgen.ppt.Presentation` | `mlreportgen.report.Report`

**Introduced in R2014b**

## **setedit**

Start Report Explorer

### **Syntax**

```
setedit (filename)
```

### **Description**

`setedit (filename)` opens the Report Explorer and loads the report setup file named `filename`. If a file with the specified name does not exist, Report Explorer opens an empty report setup file with that name.

### **See Also**

`rptlist` | `report` | `rptconvert`

### **Topics**

“Working with the Report Explorer” on page 1-16

# unzipDOCXTemplate

Unzip zipped DOTX template file

## Syntax

```
unzipDOCXTemplate(zippedTemplate)
unzipDOCXTemplate(zippedTemplate, unzippedTemplateFolder)
```

## Description

`unzipDOCXTemplate(zippedTemplate)` unzips the DOTX template specified by `zippedTemplate` into a subfolder of the folder that contains the zipped template. If `zippedTemplate` does not specify a file extension, the function assumes the extension is `.dotx`, the default extension for Microsoft Word template files.

`unzipDOCXTemplate(zippedTemplate, unzippedTemplateFolder)` unzips the DOTX template specified by `zippedTemplate` into the directory specified by `unzippedTemplateFolder`.

## Examples

### Unzip DOTX Template into Subfolder of Zipped Template Folder

Unzip a zipped DOTX template called `myTemplate`.

```
unzipDOCXTemplate("myTemplate.dotx");
```

### Unzip DOTX Template into Specified Folder

This example assumes that there is a zipped DOTX template called `myTemplate` in the current folder and a folder called `H:\report_templates`.

```
unzipDOCXTemplate("myTemplate.dotx", "H:\report_templates\myTemplate");
```

## Input Arguments

### **zippedTemplate** — Path for zipped DOTX template file

string scalar | character vector

Full path for the zipped DOTX template specified as a string scalar or character vector. If you do not include the file extension, the function assumes the extension is `.dotx`, the default extension for Word Template files.

### **unzippedTemplateFolder** — Folder to unzip template contents into

string scalar | character vector

The folder to unzip template contents into specified as a string scalar or character vector.

**See Also**

`mlreportgen.dom.Document.createTemplate` | `zipDOCXTemplate`

**Topics**

“Output Types and Report Generator Packages” on page 13-14

**Introduced in R2017b**



# unzipTemplate

Unzip zipped DOM template

## Syntax

```
unzipTemplate(zippedTemplatePath)
unzipTemplate(zippedTemplatePath,unzippedTemplatePath)
```

## Description

`unzipTemplate(zippedTemplatePath)` unzips the DOM template zip file specified by `zippedTemplatePath` into a subfolder of the folder that contains the zipped template.

`unzipTemplate(zippedTemplatePath,unzippedTemplatePath)` unzips the DOM template into the folder specified by `unzippedTemplatePath`.

## Examples

### Unzip DOM Template into Subfolder of Zipped Template Folder

Unzip a zipped DOM template called `myTemplate`.

```
unzipTemplate('myTemplate');
```

### Unzip DOM Template into Specified Folder

This example assumes that there is a zipped DOM template called `myTemplate` in the current folder and a folder called `H:\report_templates`.

```
unzipTemplate('myTemplate.htmxx','H:\report_templates\myTemplate');
```

## Input Arguments

### **zippedTemplatePath** — Path of the zipped DOM template

character vector

If you do not include a file extension in the path, the function assumes the extension is `.htmxx`.

If you do not use the `unzippedTemplatePath` argument, the `unzipTemplate` function unzips the template into a subfolder of the folder that contains the zipped template. The name of the unzipped template folder is the same as the root name of the zipped template. The root name is the zipped template name without its file extension.

### **unzippedTemplatePath** — The location to store the unzipped DOM template

character vector

The template is unzipped into the folder that you specify in `unzippedTemplatePath`.

**See Also**

`mlreportgen.dom.Document.createTemplate | zipTemplate`

**Topics**

“Output Types and Report Generator Packages” on page 13-14

**Introduced in R2014b**

# zipDOCXTemplate

Package unzipped DOTX template into DOTX file

## Syntax

```
zipDOCXTemplate(unzippedTemplateFolder)
zipDOCXTemplate(zippedTemplate, unzippedTemplateFolder)
```

## Description

`zipDOCXTemplate(unzippedTemplateFolder)` zips the unzipped template files specified by the directory `unzippedTemplateFolder`. The resulting zipped template file name is the name specified in `unzippedTemplateFolder`, plus the file extension `.dotx`. The zipped template folder structure duplicates the folder structure of the unzipped template. The file names in the unzipped template must contain only ASCII characters. The unzipped directory must contain a valid DOTX template package structure based on OOXML standard.

`zipDOCXTemplate(zippedTemplate, unzippedTemplateFolder)` packages the unzipped DOTX template into the file specified by `zippedTemplate`. The resulting compressed template preserves the directory structure of the uncompressed template. The `zipDOCXTemplate` function zips all of the files in the unzipped template folder, including files in subfolders.

## Examples

### Package Specified Folder into a DOTX File

Package the template in the folder `myTemplate` into a DOTX file called `myTemplate.dotx`.

```
zipDOCXTemplate("myTemplate");
```

### Package a Template with a Specified Name

Package the template in the folder `myTemplate` into a DOTX file called `myReportTemplate.dotx`.

```
zipDOCXTemplate("myReportTemplate.dotx", "myTemplate");
```

## Input Arguments

### **unzippedTemplateFolder** — Path for unzipped template files

string scalar | character vector

Path to folder containing unzipped template files, specified as a string scalar or character vector.

### **zippedTemplate** — Path for packaged DOTX template file

string scalar | character vector

Full path for the packaged DOTX template specified as a string scalar or character vector.

**See Also**

`mlreportgen.dom.Document.createTemplate` | `unzipDOCXTemplate`

**Topics**

“Output Types and Report Generator Packages” on page 13-14

**Introduced in R2017b**

# zipTemplate

Package DOM HTML and PDF template in zip file

## Syntax

```
zipTemplate(unzippedTemplateFolder)
zipTemplate(zippedTemplate,unzippedTemplateFolder)
zipTemplate(zippedTemplate,unzippedTemplateFolder,mainDocument)
zipTemplate(zippedTemplate,unzippedTemplateFolder,mainDocument,partTemplates)
```

## Description

`zipTemplate(unzippedTemplateFolder)` zips (compresses and puts in a zip file) the unzipped DOM template in `unzippedTemplateFolder`. The resulting zipped template file name is the name specified in `unzippedTemplateFolder`, plus the file extension `htm` or `pdf`. The `zipTemplate` function zips all of the files in the unzipped template folder, including files in subfolders. The zipped template folder structure duplicates the folder structure of the unzipped template. The file names in the unzipped template must contain only ASCII characters.

Use this syntax if you created the unzipped template by unzipping a template created in any of these ways:

- Used `mlreportgen.dom.Document.createTemplate`
- Copied the template from a default DOM template
- Created the template without using the DOM API or DOM templates and the zipped file complies with the conditions listed in “Tips”.

`zipTemplate(zippedTemplate,unzippedTemplateFolder)` zips the unzipped DOM template into the file specified by `zippedTemplate`.

`zipTemplate(zippedTemplate,unzippedTemplateFolder,mainDocument)` zips the unzipped DOM template into the file specified by `zippedTemplate`. Use the `mainDocument` argument to specify the name of main document in the unzipped template if the main document name in the unzipped template is not `report.html` or `root.html` and your document part template library file, if it exists, is in a file called `docpart_templates.html`.

`zipTemplate(zippedTemplate,unzippedTemplateFolder,mainDocument,partTemplates)` zips the unzipped DOM template into the file specified by `zippedTemplate`. Use this syntax when the unzipped template includes a document part template library file whose file name is not `docpart_templates.html`. You must specify `mainDocument` as the third argument, even if the main document file is called `report.html` or `root.html`.

## Examples

### Zip to a File Whose Base Name is the Unzipped Template Folder

Zip the template `myTemplate` into a zip file called `myTemplate.htm`.

```
zipTemplate('myTemplate');
```

### Zip to a Specified Zip File Name

Zip the template `myTemplate` into a zip file called `myReportTemplate.htmx`.

```
zipTemplate('myReportTemplate.htmx', 'myTemplate');
```

### Zip When Main Document and Part Template File Use Custom Names

Zip a template whose main part is `mainpart.html` and whose part template library file is `documentpart_templates.html`.

```
zipTemplate('myTemplate.htmx', 'myTemplate', ...  
    'mainpart.html', 'documentpart_templates.html');
```

## Input Arguments

### **unzippedTemplateFolder** — Path to folder containing unzipped template

character vector

Path to folder containing unzipped template, specified as a character vector. The file names in the unzipped template must contain only ASCII characters.

### **zippedTemplate** — Path for zipped DOM template file

character vector

Full path for the zipped DOM template, including the file extension `.htmxtx` or `.pdfxtx`, specified as a character vector.

### **mainDocument** — Name of main document file

character vector

Main document file name, including the file extension, specified as a character vector.

### **partTemplates** — Document part library file name

character vector

Document part library file name, including the file extension, specified as a character vector.

## Tips

- If you created the unzipped template by unzipping a template created by using `mlreportgen.dom.Document.createTemplate` or copying the template from a default DOM template, you can use either of these syntaxes with no further action:

```
zipTemplate(unzippedTemplateFolder)
```

```
zipTemplate(zippedTemplate, unzippedTemplateFolder)
```

You can also use either of those two syntaxes if the unzipped template was created without using the DOM interface and the template complies with the following requirements.

- The main document file is named either `report.html` or `root.html`.
- The unzipped template either does not include a document part template library file, or it includes a document part template library file named `docpart_templates.html`.
- The unzipped template stores images in a folder named `images`.

If the unzipped template main document file is not named either `report.html` or `root.html`, use the `mainDocument` input argument.

If the unzipped template includes a document part template library file with a name other than `docpart_templates.html`, use the `partTemplates` input argument.

If the unzipped template stores images in a folder other than one named `images` in the root folder of the template, include a text file called `_imgprefix` in the folder that contains images for the unzipped template. In the `_imgprefix` file, you can include a prefix for the DOM interface to use to generate names images appended to documents. For example, if the `_imgprefix` file contains the prefix `graphic`, the generated image names are `graphic1.png`, `graphic2.png`, and so on. If you leave the `_imgprefix` file empty, then the generated images use the prefix `image`.

## See Also

`mlreportgen.dom.Document.createTemplate` | `unzipTemplate`

## Topics

“Output Types and Report Generator Packages” on page 13-14

## Introduced in R2014b





# Classes

---

## mlreportgen.dom.AllowBreakAcrossPages class

**Package:** mlreportgen.dom

Allow row to straddle page break

### Description

Specifies whether to allow row to straddle page break. This format applies only to Word documents.

### Construction

`breakAcrossPagesObj = AllowBreakAcrossPages()` allows a row to flow onto the next page when it cannot fit entirely on the current page.

`breakAcrossPagesObj = AllowBreakAcrossPages(tf)` forces a row to start on the next page when it cannot fit on the current page and `tf = false`.

### Input Arguments

#### **tf — Allow row to flow onto next page**

`true` (default) | logical value

A setting of `false` (or 0) forces a row to start on the next page when it cannot fit on the current page. A setting of `true` (or 1) allows a row to flow onto the next page when it cannot fit entirely on the current page.

Data Types: `logical`

### Output Arguments

#### **breakAcrossPagesObj — Control table row page break**

`mlreportgen.dom.AllowBreakAcrossPages` object

Specification of table row page break handling, represented by an `mlreportgen.dom.AllowBreakAcrossPages` object.

### Properties

#### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

#### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the

object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Value — Allow row to flow onto next page**

true (default) | logical value

The possible values are:

- 0— forces a row to start on the next page when it cannot fit on the current page
- 1— allows a row to flow onto the next page when it cannot fit entirely on the current page

Data Types: `logical`

**See Also**

`mlreportgen.dom.TableRow` | `mlreportgen.dom.RepeatAsHeaderRow`

**Topics**

“Report Formatting Approaches” on page 13-17

## mlreportgen.dom.AutoNumber class

**Package:** mlreportgen.dom

Automatically generated number

### Description

Automatically generated number for a DOM document element object.

### Construction

`autoObj = AutoNumber()` creates an automatically generated number without a specified number stream.

`autoObj = AutoNumber(stream)` creates a number based on the specified numbering stream.

`autoObj = AutoNumber(stream, styleName)` creates a number using the specified style.

### Input Arguments

#### **stream — Numbering stream for generating the number**

character vector

Specify a numbering stream, using the value of the `mlreportgen.dom.AutoNumberStream` object `StreamName` property.

If the specified stream does not exist, the DOM interface creates an Arabic number stream having the specified name with an initial value of 0. To use a stream with other properties, such as Roman numerals, create a stream using `mlreportgen.dom.Document.createAutoNumberStream`.

#### **styleName — Name of number style defined in the template**

character vector

Name of number style defined in the template, specified as a character vector. The style specified by `styleName` must be defined in the template used to create the document to which the number is appended.

### Output Arguments

#### **autoObj — Automatically created number object**

`mlreportgen.dom.AutoNumber` object

Automatically created number object, specified as an `mlreportgen.dom.AutoNumber` object.

### Properties

#### **BackgroundColor — Background color**

[] (default) | character vector | string scalar

Background color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `BackgroundColor` property adds a corresponding `mlreportgen.dom.BackgroundColor` format object to the `Style` property. Setting the `BackgroundColor` property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

### **Bold — Option to use bold for number**

[] (default) | logical value

To make text bold, set this property to `true` or `1`. If this property is empty and the `StyleName` property for this document element specifies a style sheet style, the weight of the number is determined by that style. Setting the `Bold` property adds a corresponding `mlreportGen.dom.Bold` format object to the `Style` property of this document element. Removing the `Bold` property setting removes the object.

Data Types: logical

### **Color — Text color**

[] (default) | character vector | string scalar

Text color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `Color` property adds a corresponding `mlreportgen.dom.Color` format object to the `Style` property. Setting the `Color` property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

### **CustomAttributes — Custom attributes of document element**

array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this document element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. The custom attributes must be supported by the output format.

### **FontFamilyName — Name of font family for text**

[] | character vector | string scalar

Name of a font family to use for text, specified as a character vector or string scalar.

Setting the `FontFamilyName` property adds a corresponding `mlreportgen.dom.FontFamily` format object to the `Style` property. Setting the `FontFamilyName` property is the same as setting the `FontName` property of an `mlreportgen.dom.FontFamily` object. Setting the `FontFamilyName` property to an empty value removes the object.

To specify substitutions for the font, do not specify the font family by setting this property. Instead create and add an `mlreportgen.dom.FontFamily` object to the `Style` property.

Example: 'Courier New'

### FontSize — Font size

character vector

If you need to specify substitutions for this font, do not set this property. Instead create and add an `mlreportgen.dom.FontFamily` object to the `Style` property of this document element.

Setting the `FontSize` property adds a corresponding `mlreportGen.dom.FontSize` format object to the `Style` property for this document element. Removing the `FontSize` property setting removes the object.

Font size, specified in the form `valueUnits`, where `Units` is an abbreviation for the units. Use one of these abbreviations for the units.

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Italic — Option to use italics for a number

[] (default) | logical value

To use italics for a number, set this property to `true`. If this property is empty and the `StyleName` property for this document element specifies a style sheet style, the slant of the number is determined by that style. Setting the `Italic` property adds a corresponding `mlreportGen.dom.Italic` format object to the `Style` property of this document element. Removing the `Italic` property setting removes the object.

Data Types: logical

### Strike — Text strikethrough

[] (default) | 'none' | 'single' | 'double'

Text strikethrough, specified as one of these values:

- `'none'` — No strikethrough
- `'single'` — Single line
- `'double'` — Double line (for Word reports only)

Setting the `Strike` property adds a corresponding `mlreportgen.dom.Strike` format object to the `Style` property for this document element. Setting the `Strike` property to an empty value removes the object.

**Style – Formats that define the element style**

array of format objects

The formats specified by this property override corresponding formats defined by the style sheet style specified by the `StyleName` property of this element. Formats that do not apply to this element are ignored.

**StyleName – Style for the number**

character vector

The style specified by `styleName` must be defined in the template used to create the document element to which this number is appended.

**Tag – Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Underline – Type of underline**

[] (default) | 'single' | 'none' | ...

Type of underline, specified as one of the values in this table.

Underline value	Description	Supported Output Types
'single'	Single line	All
'words'	Underline all characters except spaces	Word
'double'	Double line	Word
'thick'	Thick line	Word
'dotted'	Dotted line	Word
'dottedHeavy'	Thick dotted line	Word
'dash'	Dashed line	Word
'dashedHeavy'	Line with heavy dashes	Word
'dashLong'	Line with long dashes	Word
'dashLongHeavy'	Line with heavy long dashes	Word
'dotDash'	Dot-dash line	Word
'dashDotHeavy'	Heavy dash-dot line	Word
'dotDotDash'	Dot-dot-dash line	Word
'dashDotDotHeavy'	Line with heavy dashes with two dots between the dashes	Word
'wave'	Wavy line	Word
'wavyHeavy'	Heavy wavy	Word

Underline value	Description	Supported Output Types
'wavyDouble'	Double wavy line	Word
'none'	No underline	All

Setting the Underline property adds a corresponding `mlreportgen.dom.Underline` format object to the Style property. Setting the Underline property to an empty value removes the object.

To specify both the color and the type of the underline, do not set the Underline property. Instead, set the Style property to include an `mlreportgen.dom.Underline` format object that specifies the underline type and color.

### WhiteSpace — How to treat white space in text

[ ] (default) | 'normal' | 'nowrap' | ...

How to treat white space in text, specified as one of the values in this table.

Value	Description	Supported Output Types
'normal'	For HTML and PDF, this value removes spaces at the beginning and the end of text. Multiple spaces in the text collapse to a single space.  For Word, this value removes spaces at the beginning and end of text.	All
'nowrap'	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
'pre'	Preserves white space. Text wraps only on line breaks. Acts like the <code>&lt;pre&gt;</code> tag in HTML.	HTML and PDF
'pre-line'	Sequences of white space collapse into a single white space. Text wraps when necessary and on line breaks.	HTML and PDF
'pre-wrap'	Preserves white space. Text wraps when necessary and on line breaks.	HTML and PDF
'preserve'	Same as 'pre'.	All

Setting the WhiteSpace property adds a corresponding `mlreportgen.dom.WhiteSpace` format object to the Style property. Setting the WhiteSpace property to an empty value removes the object.



## Methods

Method	Purpose
<b>append</b> Use <code>AutoNumber.append</code> in a similar way to how you use <code>ExternalLink.append</code> .	Append a custom element to this number.
<b>clone</b> Use <code>AutoNumber.clone</code> in a similar way to how you use <code>Paragraph.clone</code> .	Copy the number object.

## Examples

### Use Automatically Generated Numbers for Chapters and Tables

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);

p = Paragraph('Chapter ');
p.Style = {CounterInc('chapter'),CounterReset('table'),...
  WhiteSpace('preserve')};
append(p,AutoNumber('chapter'));
append(d,p);

p = Paragraph('Table ');
append(p,AutoNumber('chapter'));
append(p,'. ');
append(p,AutoNumber('table'));
p.Style = {CounterInc('table'),WhiteSpace('preserve')};
append(d,p);

p = Paragraph('Chapter ');
p.Style = {CounterInc('chapter'),CounterReset('table'),...
  WhiteSpace('preserve')};
append(p,AutoNumber('chapter'));
append(d,p);

p = Paragraph('Table ');
append(p, AutoNumber('chapter'));
append(p,'. ');
append(p,AutoNumber('table'));
p.Style = {CounterInc('table'),WhiteSpace('preserve')};
append(d,p);

close(d);
rptview('test',doctype);
```

### See Also

`createAutoNumberStream` | `getAutoNumberStream` | `mlreportgen.dom.CounterInc` | `mlreportgen.dom.CounterReset`

**Topics**

"Automatically Number Document Content" on page 13-101

# mlreportgen.dom.AutoNumberStream class

**Package:** mlreportgen.dom

Numbering stream

## Description

A numbering stream generates a sequence of numbers for numbering chapters, tables, figures, and other document objects. To create a numbering stream object, use the `createAutoNumberStream` method.

## Properties

### **CharacterCase** — Character case of generated numbers

'lower' | 'upper'

Character case of generated numbers, specified as:

- 'lower' — lowercase (for example, a, b ,c)
- 'upper' — uppercase (for example, A, B, C)

### **CharacterType** — Character type of generated numbers

'alphabetic' | 'arabic' | 'roman'

Character type of generated numbers, specified as:

- 'alphabetic' — for example, a, b, c
- 'arabic' — for example, 1, 2, 3
- 'roman' — for example, i, ii, iii

### **Id** — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **InitialValue** — Initial value of generated number

character vector

The value of this property should be one less than the number that you want to be generated first. For example, if you want the number of the first item to be numbered by this stream to be 2, set the value of this property to 1.

### **StreamName** — Name of numbering stream

character vector

Name of numbering stream, specified as a character vector.

### **Tag** — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**See Also**

`createAutoNumberStream` | `getAutoNumberStream` | `mlreportgen.dom.CounterInc` | `mlreportgen.dom.CounterReset`

**Topics**

“Automatically Number Document Content” on page 13-101

# mlreportgen.dom.BackgroundColor class

**Package:** mlreportgen.dom

Background color of document element

## Description

Specifies the background color of a document element

## Construction

`backgroundColorObj = BackgroundColor()` creates a white background.

`backgroundColorObj = BackgroundColor(colorName)` creates a background color object based on the specified CSS color name.

`backgroundColorObj = BackgroundColor(rgbValue)` creates a background color object using the hexadecimal RGB color value.

## Input Arguments

### **colorName** — Name of a color to use

character vector

The name of a color, specified as a character vector. The name must be a CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.

### **rgbValue** — Hexadecimal RGB (red, green, blue) color value

character vector

A character vector using the following RGB format: `#RRGGBB`. Use `#` as the first character and two-digit hexadecimal numbers each for the red, green, and blue values. For example, `'#0000ff'` specifies blue.

## Output Arguments

### **backgroundColorObj** — Background color

mlreportgen.dom.BackgroundColor object

Background color for a report object, returned as an mlreportgen.dom.BackgroundColor object

## Properties

### **HexValue** — Hexadecimal color value (read-only)

character vector

Hexadecimal number representing an RGB color value. For example, `'#8b008b'` specifies dark magenta. You can use either uppercase or lowercase letters as part of a hexadecimal value.

### **Id** — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Value — CSS color name or hexadecimal RGB value for this color**

character vector

Either a CSS color name or a hexadecimal RGB value, specified as a character vector.

## Examples

**Create and Apply a Background Color**

Create a deep sky blue background color object and apply it to a paragraph. Instead of specifying the CSS color name DeepSkyBlue, you can use the hexadecimal value #00bfff.

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);
blue = 'DeepSkyBlue';
% blue = '#00BFFF';
colorfulStyle = {Bold,Color(blue),BackgroundColor('Yellow')};
p = Paragraph('deep sky blue paragraph with yellow background');
p.Style = colorfulStyle;
append(d,p);
close(d);
rptview('test',doctype);
```

**See Also**

mlreportgen.dom.Color

**Topics**

“Report Formatting Approaches” on page 13-17

# mlreportgen.dom.Bold class

**Package:** mlreportgen.dom

Bold for text object

## Description

Specifies whether to use bold for a text object

The mlreportgen.dom.Bold class is a handle class.

## Creation

### Description

`boldObj = Bold()` creates a bold object that specifies to use bold for a text object.

`boldObj = Bold(value)` creates a bold object that specifies to use bold for a text object if `value` is true. Otherwise, creates a bold object that specifies to use regular weight text.

### Input Arguments

#### **value — Option to use bold or regular weight for text object**

[] (default) | logical value

A setting of `false` (or 0) uses regular weight text. A setting of `true` (or 1) renders text in bold.

Data Types: logical

## Properties

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### **Value — Option to use bold or regular weight for a text object**

[] (default) | logical value

The possible values are:

- 0— uses regular weight text
- 1— renders text in bold

Data Types: logical

### Examples

#### Create Paragraph With Bold and Regular-Weight Text

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);
p = Paragraph('Bold text ');
p.Style = {Bold};
append(d,p);

t = Text('Regular weight text');
t.Style = {Bold(false)};
append(p,t);
close(d);
rptview('test',doctype);
```

#### See Also

`mlreportgen.dom.Italic`

#### Topics

“Report Formatting Approaches” on page 13-17

**Introduced in R2014b**



# mlreportgen.dom.Border class

**Package:** mlreportgen.dom

Border properties of object

## Description

Specifies the border properties of an object.

## Construction

`borderObj = Border()` creates an unspecified border.

`borderObj = Border(style)` creates a border having the specified style.

`borderObj = Border(style,color)` creates a border having the specified style and color.

`borderObj = Border(style,color,width)` creates a border having the specified style, color, and width.

## Input Arguments

### **style** — Default style of border segments

character vector

Use one of the values in the table.

Border Value	Description	Supported Output Types
'dashed'	Dashed line	All output types
'dashdotstroked'	Line with alternating diagonal dashes and dot	Word
'dashsmallgap'	Dashed line with a small gap between dashes	Word
'dotted'	Dotted line	All output types
'dotdash'	Line with alternating dots and dashes	Word
'dotdotdash'	Line with alternating double dots and a dash	Word
'double'	Double line	All output types
'doublewave'	Double wavy line	Word
'groove'	3-D effect grooved line	HTML and PDF

Border Value	Description	Supported Output Types
'hidden'	<p>No line</p> <p>The 'none' border type also produces no line. However, conflicting borders are handled differently for 'hidden' types than for 'none' types. The 'hidden' border type takes precedence over a conflicting border type. A conflicting border type takes precedence over the 'none' type.</p>	HTML and PDF
'inset'	3-D effect line	All output types
'none'	<p>No line</p> <p>The 'hidden' border type also produces no line. However, conflicting borders are handled differently for 'hidden' types than for 'none' types. The 'hidden' border type takes precedence over a conflicting border type. A conflicting border type takes precedence over the 'none' type.</p>	All output types
'outset'	3-D effect line	All output types
'ridge'	3-D effect ridged line	HTML and PDF
'single'	Single line	Word
'solid'	Single line	HTML and PDF
'thick'	Thick line	Word
'thickthinlargegap'	Dashed line with alternating thick and thin dashes with a large gap	Word
'thickthinmediumgap'	Dashed line with alternating thick and thin dashes with a medium gap	Word
'thickthinsmallgap'	Dashed line with alternating thick and thin dashes with a small gap	Word
'thinthicklargegap'	Dashed line with alternating thin and thick dashes with a medium gap	Word
'thinthickmediumgap'	Dashed line with alternating thin and thick dashes, with a medium gap	Word

Border Value	Description	Supported Output Types
'thinthicksmallgap'	Dashed line with alternating thin and thick dashes with a small gap	Word
'thinthickthinlargegap'	Dashed line with alternating thin and thick dashes with a large gap	Word
'thinthickthinmediumgap'	Dashed line with alternating thin and thick dashes with a medium gap	Word
'thinthickthinsmallgap'	Dashed line with alternating thin and thick dashes with a small gap	Word
'threedemboss'	Embossed effect line	Word
'threedengrave'	Engraved effect line	Word
'triple'	Triple line	Word
'wave'	Wavy line	Word

**color — Color of border**

character vector

You can specify:

- The name of a color. The name must be a CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- A hexadecimal RGB (truecolor) value as #RRGGBB. For example, #0000ff is a shade of blue.

**width — Width of border**

character vector

Width of the border, specified as a character vector, in the format `valueUnits`. `Units` is an abbreviation for the units. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Output Arguments****borderObj — Table border**

mlreportgen.dom.Border object

Table border, represented by an `mlreportgen.dom.Border` object.

## Properties

### Color — Default color of border segments

character vector

You can specify:

- The name of a color. The name must be a CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- A hexadecimal RGB (truecolor) value as #RRGGBB. For example, #0000ff is a shade of blue.

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Style — Default style of border segments

character vector

For details, see the description of the `style` input argument for the `mlreportgen.dom.Border` constructor.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Width — Width of border

character vector

Width of the border, specified as a character vector in the form `valueUnits`. `Units` is an abbreviation for the units. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

### BottomColor — Bottom border segment color

character vector

Bottom border segment color, specified as a character vector.

### BottomStyle — Bottom border segment style

character vector

Bottom border segment style, specified as a character vector.

**BottomWidth — Bottom border segment width**

character vector

Bottom border segment width, specified as a character vector.

**TopColor — Top border segment color**

character vector

Top border segment color, specified as a character vector.

**TopStyle — Top border segment style**

character vector

Top border segment style, specified as a character vector.

**TopWidth — Top border segment width**

character vector

Top border segment width, specified as a character vector.

**LeftColor — Left border segment color**

character vector

Left border segment color, specified as a character vector.

**LeftStyle — Left border segment style**

character vector

Left border segment style, specified as a character vector.

**LeftWidth — Left border segment width**

character vector

Left border segment width, specified as a character vector.

**RightColor — Right border segment color**

character vector

Right border segment color, specified as a character vector.

**RightStyle — Right border segment style**

character vector

Right border segment style, specified as a character vector.

**RightWidth — Right border segment width**

character vector

Right border segment width, specified as a character vector.

## Examples

**Format Table Borders**

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);
t = Table(magic(5));
t.Style = {Border('inset','crimson','6pt'),Width('50%')};
t.TableEntriesInnerMargin = '6pt';
append(d,t);
close(d);
rptview('test',doctype);
```

**See Also**

[mlreportgen.dom.Table](#) | [mlreportgen.dom.ColSep](#) | [mlreportgen.dom.RowSep](#)

**Topics**

“Create Informal Tables” on page 13-64

# mlreportgen.dom.BorderCollapse class

**Package:** mlreportgen.dom

Collapse HTML table borders

## Description

Specifies whether to collapse table borders. This class applies only to HTML tables.

## Construction

`borderCollapseObj = BorderCollapse()` creates an unspecified format. Nothing is inserted in the generated table markup.

`borderCollapseObj = BorderCollapse(value)` creates a border collapse object having the specified value.

## Input Arguments

**value — Specify whether to collapse border**

'on' | 'off'

Setting to specify whether to collapse table borders, specified as 'on' to collapse or 'off' to leave the table and adjacent cell borders separate.

## Output Arguments

**borderCollapseObj — Specify whether to collapse table borders**

mlreportgen.dom.BorderCollapse object

Specify whether to collapse table borders, represented by an mlreportgen.dom.BorderCollapse object.

## Properties

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Value — Specify whether to collapse border**`'on' | 'off'`

Setting to specify whether to collapse table borders, specified as 'on' to collapse or 'off' to leave the table and adjacent cell borders separate.

**Examples****Collapse and Separate Table Borders**

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);

magicArray = magic(5);

p = Paragraph('Collapsed Borders');
append(d,p);
table = Table(magicArray);
table.Style = {Border('solid'),BorderCollapse('on')};
    for r = 1:5
        for c = 1:5
            table.entry(r,c).Style = {Border('solid')};
        end
    end
append(d,table);

p = Paragraph('Separate Borders');
append(d,p);
table = Table(magicArray);
table.Style = {Border('solid'),BorderCollapse('off')};
    for r = 1:5
        for c = 1:5
            table.entry(r,c).Style = {Border('solid')};
        end
    end
append(d,table);

close(d);
rptview(d.OutputPath,doctype);
```

**See Also**

[mlreportgen.dom.Border](#) | [mlreportgen.dom.TableRow](#) | [mlreportgen.dom.TableColSpec](#)  
| [mlreportgen.dom.TableEntry](#)

**Topics**

“Create Informal Tables” on page 13-64



# mlreportgen.dom.CharEntity class

**Package:** mlreportgen.dom

Create character entity reference

## Description

Create a reference to a character entity reference.

## Construction

`charEntityObj = CharEntity()` creates a reference to a nonbreaking space ( ) entity. Appending this reference to a document inserts a nonbreaking space.

`charEntityObj = CharEntity(name)` creates a reference to the character entity specified by name.

`charEntityObj = CharEntity(name,n)` creates n references to the character entity specified by name, that is, a character vector of n special characters.

## Input Arguments

### **name — Character entity name**

character vector

Entity name must be listed at [https://en.wikipedia.org/wiki/List\\_of\\_XML\\_and\\_HTML\\_character\\_entity\\_references](https://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references).

### **n — Number of character entities to use**

integer

Number of character entities to use, specified as an integer.

Data Types: uint16

## Output Arguments

### **charEntityObj — Reference to a character entity**

mlreportgen.dom.CharEntity object

Reference to a character entity, represented by an mlreportgen.dom.CharEntity object.

## Properties

### **BackgroundColor — Background color**

[] (default) | character vector | string scalar

Background color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `BackgroundColor` property adds a corresponding `mreportgen.dom.BackgroundColor` format object to the `Style` property. Setting the `BackgroundColor` property to an empty value removes the object.

Example: `'blue'`

Example: `'#0000ff'`

### **Bold — Option to use bold for number**

logical value

To make text bold, set this property to `true` or `1`. If this property is empty and the `StyleName` property for this document element specifies a style sheet style, the weight of the number is determined by that style. Setting the `Bold` property adds a corresponding `mreportgen.dom.Bold` format object to the `Style` property of this document element. Removing the `Bold` property setting removes the object.

Data Types: `logical`

### **Color — Text color**

`[]` (default) | character vector | string scalar

Text color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form `#RRGGBB`.

Setting the `Color` property adds a corresponding `mreportgen.dom.Color` format object to the `Style` property. Setting the `Color` property to an empty value removes the object.

Example: `'blue'`

Example: `'#0000ff'`

### **Content — Text contained by this document element**

character vector

Text contained by this document element, specified as a character vector.

### **CustomAttributes — Custom attributes of document element**

array of `mreportgen.dom.CustomAttribute` objects

Custom attributes of this document element, specified as an array of `mreportgen.dom.CustomAttribute` objects. The custom attributes must be supported by the output format.

### **FontFamilyName — Name of font family for text**

`[]` | character vector | string scalar

Name of a font family to use for text, specified as a character vector or string scalar.

Setting the `FontFamilyName` property adds a corresponding `mreportgen.dom.FontFamily` format object to the `Style` property. Setting the `FontFamilyName` property is the same as setting the `FontName` property of an `mreportgen.dom.FontFamily` object. Setting the `FontFamilyName` property to an empty value removes the object.

To specify substitutions for the font, do not specify the font family by setting this property. Instead create and add an `mreportgen.dom.FontFamily` object to the `Style` property.

Example: 'Courier New'

### FontSize — Font size

character vector

If you need to specify substitutions for this font, do not set this property. Instead create and add an `mlreportgen.dom.FontFamily` object to the `Style` property of this document element.

Setting the `FontSize` property adds a corresponding `mlreportGen.dom.FontSize` format object to the `Style` property for this document element. Removing the `FontSize` property setting removes the object.

Font size, specified as a character vector in the form `valueUnits`. `Units` is an abbreviation for the units. Use one of these abbreviations for the units for the font size.

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Italic — Option to use italics for number

logical value

To use italics for a number, set this property to `true`. If this property is empty and the `StyleName` property for this document element specifies a style sheet style, the slant of the number is determined by that style. Setting the `Italic` property adds a corresponding `mlreportGen.dom.Italic` format object to the `Style` property of this document element. Removing the `Italic` property setting removes the object.

Data Types: `logical`

### Name — Name of character entity

character vector

The name is a character entity listed in [https://en.wikipedia.org/wiki/List\\_of\\_XML\\_and\\_HTML\\_character\\_entity\\_references](https://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references).

Data Types: `logical`

### Repeat — Number of times to repeat character entity

numeric value

Number of times to repeat character entity, specified as a numeric value.

Data Types: `double`

**Strike — Text strikethrough**

[] (default) | 'none' | 'single' | 'double'

Text strikethrough, specified as one of these values:

- 'none' — No strikethrough
- 'single' — Single line
- 'double' — Double line (for Word reports only)

Setting the `Strike` property adds a corresponding `mreportgen.dom.Strike` format object to the `Style` property for this document element. Setting the `Strike` property to an empty value removes the object.

**Style — Number formatting**

array of format objects

An array of format objects that specifies the format for the number.

**StyleName — Style for number**

character vector

The style specified by `styleName` must be defined in the template used to create the document element to which this number is appended.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Underline — Type of underline**

[] (default) | 'single' | 'none' | ...

Type of underline, specified as one of the values in this table.

Underline value	Description	Supported Output Types
'single'	Single line	All
'words'	Underline all characters except spaces	Word
'double'	Double line	Word
'thick'	Thick line	Word
'dotted'	Dotted line	Word
'dottedHeavy'	Thick dotted line	Word
'dash'	Dashed line	Word
'dashedHeavy'	Line with heavy dashes	Word
'dashLong'	Line with long dashes	Word

Underline value	Description	Supported Output Types
'dashLongHeavy'	Line with heavy long dashes	Word
'dotDash'	Dot-dash line	Word
'dashDotHeavy'	Heavy dash-dot line	Word
'dotDotDash'	Dot-dot-dash line	Word
'dashDotDotHeavy'	Line with heavy dashes with two dots between the dashes	Word
'wave'	Wavy line	Word
'wavyHeavy'	Heavy wavy	Word
'wavyDouble'	Double wavy line	Word
'none'	No underline	All

Setting the Underline property adds a corresponding `mlreportgen.dom.Underline` format object to the Style property. Setting the Underline property to an empty value removes the object.

To specify both the color and the type of the underline, do not set the Underline property. Instead, set the Style property to include an `mlreportgen.dom.Underline` format object that specifies the underline type and color.

#### WhiteSpace — How to treat white space in text

[ ] (default) | 'normal' | 'nowrap' | ...

How to treat white space in text, specified as one of the values in this table.

Value	Description	Supported Output Types
'normal'	For HTML and PDF, this value removes spaces at the beginning and the end of text. Multiple spaces in the text collapse to a single space.  For Word, this value removes spaces at the beginning and end of text.	All
'nowrap'	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
'pre'	Preserves white space. Text wraps only on line breaks. Acts like the <code>&lt;pre&gt;</code> tag in HTML.	HTML and PDF
'pre-line'	Sequences of white space collapse into a single white space. Text wraps when necessary and on line breaks.	HTML and PDF

Value	Description	Supported Output Types
'pre-wrap'	Preserves white space. Text wraps when necessary and on line breaks.	HTML and PDF
'preserve'	Same as 'pre'.	All

Setting the `WhiteSpace` property adds a corresponding `mlreportgen.dom.WhiteSpace` format object to the `Style` property. Setting the `WhiteSpace` property to an empty value removes the object.

## Methods

Method	Purpose
<b>append</b> Use <code>CharEntity.append</code> in a similar way to how you use <code>ExternalLink.append</code> .	Append a custom element to this character entity.
<b>clone</b> Use <code>CharEntity.clone</code> in a similar way to how you use <code>Paragraph.clone</code> .	Clone this character entity.

## Examples

### Append a British Pound Sign

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);

p = Paragraph(CharEntity('pound'));
append(d,p);
append(p,'3');

close(d);
rptview('test',doctype);
```

### Append Two Nonbreaking Spaces

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);

p = Paragraph('Some text');
append(d,p);
ce = CharEntity('nbsp',5);
append(p,ce);
append(p,'more text after five blank spaces');
```

```
close(d);  
rptview('test',doctype);
```

## **See Also**

[mlreportgen.dom.Text](#) | [mlreportgen.dom.Paragraph](#)

## **Topics**

“Report Formatting Approaches” on page 13-17

## mlreportgen.dom.Color class

**Package:** mlreportgen.dom mlreportgen.dom mlreportgen.dom

Color of document element

### Description

Specifies the color of a document element.

### Construction

`colorObj = mlreportgen.dom.Color()` creates a black color object.

`colorObj = mlreportgen.dom.Color(colorName)` creates a color object based on the specified CSS color name.

`colorObj = mlreportgen.dom.Color(rgbValue)` creates a color object using the hexadecimal RGB color value.

### Input Arguments

#### **colorName** — Name of color

character vector

Name of a color. The name must be a CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.

#### **rgbValue** — Hexadecimal RGB (red, green, blue) color value

character vector

A character vector using the following RGB format: #RRGGBB. Use # as the first character and two-digit hexadecimal numbers each for the red, green, and blue values. For example, '#0000ff' specifies blue.

### Output Arguments

#### **colorObj** — Color for document element

mlreportgen.dom.Color object

Color for document element, represented by an mlreportgen.dom.Color object.

### Properties

#### **HexValue** — hexadecimal color value (read-only)

character vector

Hexadecimal number representing an RGB color value. For example, '#8b008b' specifies dark magenta. You can use either uppercase or lowercase letters as part of a hexadecimal value.

#### **Id** — ID for this document element

character vector | string scalar



ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Value — CSS color name or hexadecimal RGB value for this color

character vector

Either a CSS color name or a hexadecimal RGB value.

## Methods

### Public Methods

Method	Purpose
<code>mlreportgen.dom.Color.isValid</code>	Use this method to check if a string is a valid color.  <pre>tf = mlreportgen.dom.Color.isValid('red');</pre> <pre>tf =</pre> <pre>    logical</pre> <pre>     1</pre>
<code>mlreportgen.dom.Color.getUnitVector</code>	Use this method to get the unit vector values for a valid color.  <pre>redVector = mlreportgen.dom.Color.getUnitVector('red');</pre> <pre>redVector =</pre> <pre>     1     0     0</pre>

## Examples

### Create and Apply a Color Object

Create a blue color object and apply it to a paragraph. Instead of specifying the CSS color name 'blue', you could use the hexadecimal value '#0000ff'.

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);
```

```
colorfulStyle = {Bold,Color('blue')};  
p = Paragraph('deep sky blue paragraph');  
p.Style = colorfulStyle;  
append(d,p);  
  
close(d);  
rptview('test',doctype);
```

### See Also

`mlreportgen.dom.BackgroundColor`

### Topics

“Report Formatting Approaches” on page 13-17

## mlreportgen.dom.ColSep class

**Package:** mlreportgen.dom

Draw lines between table columns

### Description

Draw lines between table columns.

### Construction

`colSepObj = ColSep()` creates unspecified column separators.

`colSepObj = ColSep(style)` creates a column separator of the specified style.

`colSepObj = ColSep(style,color)` creates a column separator having the specified style and color.

`colSepObj = ColSep(style,color,width)` creates a column separator having the specified style, color, and width.

### Input Arguments

**style** — Style of column separator in table

character vector

Style of the table column separator, specified as one of these values.

Value	Applies To	
	DOCX	HTML and PDF
'dashed'	✓	✓
'dashdotstroked'	✓	
'dashsmallgap'	✓	
'dotted'	✓	✓
'dotdash'	✓	
'dotdotdash'	✓	
'double'	✓	✓
'doublewave'	✓	
'inset'	✓	✓

Value	Applies To	
	DOCX	HTML and PDF
'none'	✓	✓
'outset'	✓	✓
'single'	✓	
'solid'		✓
'thick'	✓	
'thickthinlargegap'	✓	
'thickthinmediumgap'	✓	
'thickthinsmallgap'	✓	
'thinthicklargegap'	✓	
'thinthickmediumgap'	✓	
'thinthicksmallgap'	✓	
'thinthickthinlargegap'	✓	
'thinthickthinmediumgap'	✓	
'thinthickthinsmallgap'	✓	
'threedemboss'	✓	
'threedengrave'	✓	
'triple'	✓	
'wave'	✓	

**color — Color of column separator in table**

character vector

You can specify:

- The name of a color. The name must be a CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- A hexadecimal RGB (truecolor) value as #RRGGBB. For example, #0000ff is a shade of blue.

**width — Width of column separator in the table**

character vector

Color of the table column separator, in the format `valueUnits`. `Units` is an abbreviation for the units. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

## Output Arguments

### colSepObj — Column separator definition

mlreportgen.dom.ColSpec object

Column separator definition, represented by an mlreportgen.dom.ColSpec object.

## Properties

### Color — Separator color

character vector

You can specify:

- The name of a color. The name must be a CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- A hexadecimal RGB (truecolor) value as #RRGGBB. For example, #0000ff is a shade of blue.

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Style — Format for separator

array of format objects

Array of format objects (such as Bold objects) that specify the format for the separator.

This property overrides corresponding formats defined by the style sheet style specified by the StyleName property.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Width — Separator width

character vector

Separator width as a percentage, for example, '100%', or a number of units of measurement in the form `valueUnits`. `Units` is an abbreviation for the units. Use one of these abbreviations for the units of a width.

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

Data Types: `char`

## Examples

### Specify Table Column Separator

This example creates table and sets the border, column separator, and row separator styles. The `TableEntriesStyle` property formats the table entries.

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);
t = Table(magic(5));

t.Style = { ...
    RowHeight('0.75in'), ...
    Border('solid','Green','6pt'), ...
    ColSep('double','DarkGreen','3pt'), ...
    RowSep('single','DarkGreen')};

t.TableEntriesStyle = { ...
    Width('0.75in'), ...
    InnerMargin('0'), ...
    OuterMargin('0'), ...
    HAlign('center'), ...
    VAlign('middle') };

append(d,t);
close(d);
rptview(d.OutputPath);
```

## See Also

`mlreportgen.dom.RowSep` | `mlreportgen.dom.Table`

## Topics

“Create Informal Tables” on page 13-64

# mlreportgen.dom.Container class

**Package:** mlreportgen.dom

Container of document objects

## Description

Creates a container element. Use the `mlreportgen.dom.Container.append` method to append document elements to the container. Use an `mlreportgen.dom.Container` object in a report to apply formats to all of the children of the container.

In HTML output, a `Container` object generates an HTML element of the type specified by its `HTMLTag` property and containing HTML elements corresponding to its DOM contents. For example, a `Container` object with the `HTMLTag` property `div` and that contains the text `Hello World` generates this markup:

```
<div><p><span>Hello World</span></p></div>
```

The generated HTML container element has the class and style properties specified by the `Container` object `StyleName` and `Style` properties, respectively. The rules of HTML CSS format inheritance assure that the generated children of the `Container` object inherit the formats specified by the `Container` object `Style` and `StyleName` properties. For example, if the `Container` object specifies red as its text color and none of its text children specify a color, the text children are colored red.

For Microsoft Word and PDF report output, a `Container` object simulates container format inheritance, applying the formats specified by the `Container` object `Style` attribute to each child, unless overridden by the child, and then appending the child to the output. Word and PDF output ignore the `HTMLTag` and `StyleName` properties of the `Container` object.

---

**Tip** You can use `mlreportgen.dom.Container` or `mlreportgen.dom.Group` objects to produce collections of document elements.

- Use a container object to apply format inheritance to a set of objects and to create HTML container elements not otherwise supported by the DOM, such as `div`, `section`, and `article`.
- Use a group object to append the same content in multiple places in a document without cloning the group.

---

## Construction

`containerObj = Container()` creates a container with an HTML tag name `div`.

`containerObj = Container(HTMLtag)` creates a container with the specified HTML tag name (for example, `div`, `section`, or `article`).

## Input Arguments

**HTMLtag — HTML container tag name**

character vector

HTML container tag name, specified as a character vector. The name must be an HTML element, such as 'div', 'section', or 'article'.

---

**Note** Word output ignores the HTML container tag.

---

## Output Arguments

### **containerObj** – Container of document objects

`mlreportgen.dom.Container` object

Container of document objects, returned as an `mlreportgen.dom.Container` object.

## Properties

### **Id** – ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **Children** – Children of container

cell array of `mlreportgen.dom.Element` objects

This read-only property lists child elements that the container contains.

### **HTMLTag** – HTML tag name of container

character vector

HTML container tag name, specified as a character vector. The name must be an HTML element, such as 'div', 'section', or 'article'.

---

**Note** Word output ignores the HTML container tag.

---

### **Parent** – Parent of document element

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

### **Style** – Format specification

array of format objects

Format specification, specified as an array of format objects. The formats specified by this property override corresponding formats defined by the style sheet style specified by the `StyleName` property of this element. Formats that do not apply to this element are ignored.

### **StyleName** – Style name

character vector

Style name, specified as a character vector. The style name is the name of a style specified in the style sheet of the document or document part to which this element is appended. The specified style



defines the appearance of this element in the output document where not overridden by the formats specified by the `Style` property of this element.

---

**Note** Word output ignores the style name.

---

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Methods

<code>append</code>	Append DOM object to container
<code>clone</code>	Copy container object

## Examples

### Create Container for Word Report Formatting

Create a container object. Word output ignores the HTML container element tag (in this example, the `div` tag).

```
import mlreportgen.dom.*;
rpt = Document('MyReport', 'docx');

c = Container();
```

Color all of the container text red.

```
c.Style = {Color('red')};
```

Append content to the container and append the container to the report.

```
append(c, Paragraph('Hello'));
append(c, Table(magic(5)));
append(rpt, c);
```

Close and generate the report.

```
close(rpt);
rptview(rpt.OutputPath);
```

### See Also

`mlreportgen.dom.Group`

**Topics**

“Add Content as a Group” on page 13-12

**Introduced in R2015a**

# mlreportgen.dom.CoreProperties class

**Package:** mlreportgen.dom

OPC core properties of document or template

## Description

OPC core properties of a document or template.

## Construction

`corePropsObj = CoreProperties()` creates an empty core properties object. Core properties are metadata stored in a document OPC package that describe various properties of the document. Windows Explorer displays some of the core properties when you select a document.

## Output Arguments

**corePropsObj** — OPC core properties

mlreportgen.dom.CoreProperties object

OPC core properties, represented by an mlreportgen.dom.CoreProperties object.

## Properties

**Category** — Category of document

character vector

Category of a document, specified as a character vector.

**ContentStatus** — Content status of document

character vector

Content status of a document, specified as a character vector.

**Created** — Creation date and time of the document

character vector

Creation date and time of the document, specified as a character vector. The format of the date and time is `yyyy-mm-dd hh:mm:ss`.

**Creator** — Creator of document

character vector

Creator of a document, specified as a character vector.

**Description** — Description of document

character vector

Description of a document, specified as a character vector.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Identifier — Identifier for document**

character vector

Identifier for a document, specified as a character vector.

**Keywords — Keywords associated with document**

array of character vectors

Keywords associated with a document, specified as a character vector.

**Language — Language of document**

character vector

Language of a document, specified as a character vector.

**LastModifiedBy — Agent that last modified this document**

character vector

Agent that last modified this document, specified as a character vector.

**LastPrinted — Last date and time this document was printed**

character vector

Last date and time this document was printed, specified as a character vector. The format of the date and time is yyyy-mm-dd hh:mm:ss.

**Modified — Last date and time this document was modified**

character vector

Last date and time this document was modified, specified as a character vector. The format of the date and time is yyyy-mm-dd hh:mm:ss.

**Revision — Revision of document**

character vector

Revision of a document, specified as a character vector.

**Subject — Subject of document**

character vector

Subject of a document, specified as a character vector.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the

object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Title – Title of document**

character vector

Title of a document, specified as a character vector.

**Version – Version of document**

character vector

Version of a document, specified as a character vector.

**See Also**

`mlreportgen.dom.Document.getCoreProperties` |  
`mlreportgen.dom.Document.setCoreProperties`

**Topics**

“Output Types and Report Generator Packages” on page 13-14

## mlreportgen.dom.CounterInc class

**Package:** mlreportgen.dom

Number stream counter incrementers

### Description

Create numbering stream counter incrementers.

### Construction

`counterIncObj = CounterInc()` creates an empty counter incrementer.

`counterIncObj = CounterInc(streamNames)` creates a counter incrementer for each specified numbering streams. Assigning this format to the style of a DOM object causes the associated stream counters to be incremented when the object is appended to a document.

### Input Arguments

#### **streamNames — Numbering stream names**

character vector

Numbering stream names, specified as a character vector. To specify multiple stream names, add a space between the stream names.

### Output Arguments

#### **counterIncObj — Numbering stream counter incrementers**

mlreportgen.dom.CounterInc objects

Numbering stream counter incrementers, represented by mlreportgen.dom.CounterInc objects.

### Properties

#### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

#### **StreamNames — Numbering stream names**

character vector

Numbering stream names, specified as a character vector. To specify multiple stream names, add a space between the stream names.

#### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Examples

### Increment Chapter Numbering

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);

p = Paragraph('Chapter ');
p.Style = {CounterInc('chapter'),WhiteSpace('preserve')};
append(p,AutoNumber('chapter'));
append(d,p);

p = Paragraph('Chapter ');
p.Style = {CounterInc('chapter'), WhiteSpace('preserve')};
append(p,AutoNumber('chapter'));
append(d,p);

close(d);
rptview('test',doctype);
```

### Specify Multiple Streams for CounterInc and CounterReset

```
import mlreportgen.dom.*
rpt = Document('MyReport','docx');

chapterStyle = {WhiteSpace('pre'), ...
    CounterReset('table figure'), ...
    CounterInc('chapter') ...
};

topicChapterStyle = {WhiteSpace('pre'), ...
    CounterReset('table figure'), ...
    CounterInc('chapter topic') ...
};

figureStyle = {WhiteSpace('pre'), ...
    CounterInc('figure'),Italic(true)};

chapter = Heading(1,'Chapter: ');
chapter.Style = chapterStyle;
append(chapter,AutoNumber('chapter'));
append(chapter,' Introduction to number streams. ');
append(rpt, chapter);

image = append(rpt,Image(which('b747.jpg')));
image.Width = '2in';
image.Height = '2in';
```

```
para = append(rpt, Paragraph('Figure '));
para.Style = figureStyle;
append(para, AutoNumber('chapter'));
append(para, '.');
append(para, AutoNumber('figure'));

image = append(rpt,Image(which('ngc6543a.jpg')));
image.Width = '2in';
image.Height = '2in';
para = append(rpt,Paragraph('Figure '));
para.Style = figureStyle;
append(para, AutoNumber('chapter'));
append(para, '.');
append(para, AutoNumber('figure'));

chapter = Heading(1, 'Chapter: ');
chapter.Style = topicChapterStyle;
append(chapter, AutoNumber('chapter'));
append(chapter, ' Topic: ');
append(chapter,AutoNumber('topic'));
append(chapter, ' How to reset and increment streams. ');
append(rpt,chapter);

image = append(rpt,Image(which('b747.jpg')));
image.Width = '2in';
image.Height = '2in';
para = append(rpt,Paragraph('Figure '));
para.Style = figureStyle;
append(para, AutoNumber('chapter'));
append(para, '.');
append(para, AutoNumber('figure'));

close(rpt);
rptview(rpt.OutputPath);
```

## See Also

[mlreportgen.dom.CounterReset](#) | [mlreportgen.dom.AutoNumber](#) | [mlreportgen.dom.AutoNumberStream](#)

## Topics

“Automatically Number Document Content” on page 13-101



# mlreportgen.dom.CounterReset class

**Package:** mlreportgen.dom

Reset numbering stream counters

## Description

Reset numbering stream counters.

## Construction

`counterResetObj = CounterReset()` creates an empty counter reset object.

`counterResetObj = CounterReset(streamNames)` creates a counter resetter for each specified numbering streams. Assigning this format to the style of a DOM object causes the associated stream counters to be reset to their initial values when the object is appended to a document.

## Input Arguments

### **streamNames — Numbering stream names**

character vector

Numbering stream names, specified as a character vector. To specify multiple stream names, add a space between the stream names.

## Output Arguments

### **reset — Numbering stream counter resets**

mlreportgen.dom.CounterReset objects

Numbering stream counter resets, represented by mlreportgen.dom.CounterReset objects.

## Properties

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **StreamName — Numbering stream names**

character vector

Numbering stream name, specified as a character vector. To specify multiple stream names, add a space between the stream names.

### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Examples

### Reset Numbering for Chapters and Tables

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);

p = Paragraph('Chapter ');
p.Style = {CounterInc('chapter'),CounterReset('table'),...
    WhiteSpace('preserve') };
append(p,AutoNumber('chapter'));
append(d,p);

p = Paragraph('Table ');
append(p,AutoNumber('chapter'));
append(p, '. ');
append(p,AutoNumber('table'));
p.Style = {CounterInc('table'),WhiteSpace('preserve') };
append(d,p);

p = Paragraph('Chapter ');
p.Style = {CounterInc('chapter'),CounterReset('table'),...
    WhiteSpace('preserve')};
append(p,AutoNumber('chapter'));
append(d,p);

p = Paragraph('Table ');
append(p,AutoNumber('chapter'));
append(p, '. ');
append(p, AutoNumber('table'));
p.Style = {CounterInc('table'),WhiteSpace('preserve')};
append(d,p);

close(d);
rptview('test',doctype);
```

### Specify Multiple Streams for CounterInc and CounterReset

```
import mlreportgen.dom.*
rpt = Document('MyReport','docx');

chapterStyle = {WhiteSpace('pre'), ...
    CounterReset('table figure'), ...
    CounterInc('chapter') ...
};

topicChapterStyle = {WhiteSpace('pre'), ...
```

```

    CounterReset('table figure'), ...
    CounterInc('chapter topic') ...
};

figureStyle = {WhiteSpace('pre'), ...
    CounterInc('figure'),Italic(true)};

chapter = Heading(1,'Chapter: ');
chapter.Style = chapterStyle;
append(chapter,AutoNumber('chapter'));
append(chapter,' Introduction to number streams.');
```

append(rpt, chapter);

```

image = append(rpt,Image(which('b747.jpg')));
image.Width = '2in';
image.Height = '2in';
para = append(rpt, Paragraph('Figure '));
para.Style = figureStyle;
append(para, AutoNumber('chapter'));
append(para, '.');
```

append(para, AutoNumber('figure'));

```

image = append(rpt,Image(which('ngc6543a.jpg')));
image.Width = '2in';
image.Height = '2in';
para = append(rpt,Paragraph('Figure '));
para.Style = figureStyle;
append(para, AutoNumber('chapter'));
append(para, '.');
```

append(para, AutoNumber('figure'));

```

chapter = Heading(1, 'Chapter: ');
chapter.Style = topicChapterStyle;
append(chapter, AutoNumber('chapter'));
append(chapter, ' Topic: ');
append(chapter,AutoNumber('topic'));
append(chapter, ' How to reset and increment streams.');
```

append(rpt,chapter);

```

image = append(rpt,Image(which('b747.jpg')));
image.Width = '2in';
image.Height = '2in';
para = append(rpt,Paragraph('Figure '));
para.Style = figureStyle;
append(para, AutoNumber('chapter'));
append(para, '.');
```

append(para, AutoNumber('figure'));

```

close(rpt);
rptview(rpt.OutputPath);
```

## See Also

mlreportgen.dom.CounterInc | mlreportgen.dom.AutoNumber |  
mlreportgen.dom.AutoNumberStream

**Topics**  
"Automatically Number Document Content" on page 13-101

# mlreportgen.dom.CSSProperties class

**Package:** mlreportgen.dom

Array of CSS properties for formatting HTML output

## Description

Creates an object that contains one or more cascading style sheet (CSS) formats for HTML output. Specify the formats using mlreportgen.dom.CSSProperty. For information on CSS properties, see W3Schools.com/cssref.

---

**Note** Use CSSProperty objects only for formats not supported by DOM format objects. Most DOM format objects work for any output type. Using CSSProperty objects makes your report application specific to HTML output.

---

## Construction

props = CSSProperties(prop) creates a CSSProperties object based on mlreportgen.dom.CSSProperty objects. The mlreportgen.dom.CSSProperty object specifies the CSS format and value.

## Input Arguments

### prop — CSS property

mlreportgen.dom.CSSProperty object | array of mlreportgen.dom.CSSProperty objects | cell array of mlreportgen.dom.CSSProperty objects

CSS property format, specified as an mlreportgen.dom.CSSProperty object or as an array or cell array of mlreportgen.dom.CSSProperty objects.

## Output Arguments

### props — CSS properties

mlreportgen.dom.CSSProperties object

CSS properties, returned as an mlreportgen.dom.CSSProperties object.

## Properties

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Parent — Parent of document element

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

### Properties — CSS properties

array of `mlreportgen.dom.CSSProperty` objects

CSS properties, specified as an array of `mlreportgen.dom.CSSProperty` objects.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Examples

### Apply CSS Properties to a List

This example uses a `CSSProperties` object to apply an HTML-specific list format, `list-style-position`, for which there is no DOM equivalent.

```
import mlreportgen.dom.*

d = Document('list-style-pos', 'html');

p = Paragraph('This list has list-style-position set to inside:');
append(d,p);
list = UnorderedList({'Earl Grey', 'Jasmine', 'Honeybush'});
list.Style = {CSSProperties(CSSProperty('list-style-position', 'inside'))};
append(d,list);

p = Paragraph('This list has list-style-position set to outside:');
append(d,p);
list = clone(list);
listprop = CSSProperty('list-style-position', 'outside');
list.Style = {CSSProperties(listprop)};
append(d,list);

close(d);
rptview(d.OutputPath);
```

### See Also

`mlreportgen.dom.F0Properties` | `mlreportgen.dom.CSSProperty`

### Topics

“Report Formatting Approaches” on page 13-17

### External Websites

W3Schools.com

**Introduced in R2016a**

## mlreportgen.dom.CSSProperty class

**Package:** mlreportgen.dom

CSS property object for formatting HTML output

### Description

Create a format object that specifies a cascading style sheet (CSS) property and value. Use the format object with `mlreportgen.dom.CSSProperties` to apply CSS properties to objects for HTML output. For information on CSS properties, see [W3Schools.com/cssref/](http://W3Schools.com/cssref/).

---

**Note** Use `CSSProperty` objects only for formats not supported by DOM format object. Most DOM format objects work for any output type. Using `CSSProperty` objects makes your report application specific to HTML output.

---

### Construction

`prop = CSSProperty(Name, Value)` creates a CSS format property that has the specified name and value.

### Input Arguments

**Name — CSS property name**

character vector

CSS property name, specified as a character vector.

**Value — Property value**

character vector

Property value for the corresponding property name, specified as a character vector.

### Output Arguments

**prop — CSS format object**

mlreportgen.dom.CSSproperty object

CSS format object, returned as an `mlreportgen.dom.CSSProperty` object.

### Properties

**Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**Id — ID for this document element**

character vector | string scalar



ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **Name — CSS property name**

character vector

CSS property name, specified as a character vector.

### **Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### **Value — Property value**

character vector

Property value for corresponding property name, specified as a character vector.

## **Examples**

### **Create and Use a CSSProperty Object**

Create the CSSProperty objects `inlist` and `outlist`. When you create a list, you can use the CSSProperty object as a value to CSSProperties on the list.

```
import mlreportgen.dom.*

d = Document('List Styles', 'html');

inlist = CSSProperty('list-style-position', 'inside');
outlist = CSSProperty('list-style-position', 'outside');
p = Paragraph('The following list has list-style-position set to inside:');
append(d,p);

list = UnorderedList({'Earl Grey', 'Jasmine', 'Honeybush'});
list.Style = {CSSProperties(inlist)};
append(d,list);

p = Paragraph('The following list has list-style-position set to outside:');
append(d,p);
list = clone(list);
list.Style = {CSSProperties(outlist)};
append(d,list);
```

```
close(d);  
rptview(d.OutputPath);
```

### **See Also**

[mlreportgen.dom.CSSProperties](#) | [mlreportgen.dom.FOProperties](#)

### **Topics**

“Report Formatting Approaches” on page 13-17

### **External Websites**

[W3Schools.com/cssref/](http://W3Schools.com/cssref/)

### **Introduced in R2016a**

# mlreportgen.dom.CustomAttribute class

**Package:** mlreportgen.dom

Custom element attribute

## Description

Custom element attribute.

## Construction

`customAttributeObj = CustomAttribute()` creates an empty custom attribute.

`customAttributeObj = CustomAttribute(name)` creates an attribute having the specified name.

`customAttributeObj = CustomAttribute(name, value)` creates an attribute having the specified name and value.

## Input Arguments

### **name — Attribute name**

character vector

Attribute name, specified as a character vector.

### **value — Attribute value**

character vector

Attribute value, specified as a character vector.

## Output Arguments

### **customAttributeObj — Custom attribute**

mlreportgen.dom.CustomAttribute object

Custom attribute, represented by an mlreportgen.dom.CustomAttribute object.

## Properties

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **Name — Attribute name**

character vector

Attribute name, specified as a character vector.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Value — Value of this attribute**

character vector

Value of this attribute, specified as a character vector.

## Examples

**Create Custom Attributes for a List**

This example shows how to define custom attributes and append them to an unordered list.

```
import mlreportgen.dom.*;
d = Document('test');

ul = UnorderedList();

li = ListItem('Owl');
li.CustomAttributes = {CustomAttribute('data-animal-type','bird')};
append(ul,li);

li = ListItem('Salmon');
li.CustomAttributes = {CustomAttribute('data-animal-type','fish')};
append(ul,li);

li = ListItem('Tarantula');
li.CustomAttributes = {CustomAttribute('data-animal-type','spider')};

append(ul,li);
append(d,ul);

close(d);
rptview('test','html');
```

**See Also**

mlreportgen.dom.CustomElement | mlreportgen.dom.CustomText

# mlreportgen.dom.CustomElement class

**Package:** mlreportgen.dom

Custom element of document

## Description

Use a custom element to extend the DOM API. You can create a custom HTML or Microsoft Word element that provides functionality not yet included in the DOM API.

## Construction

`customElementObj = CustomElement()` creates an empty element.

`customElementObj = CustomElement(name)` creates a custom element having the specified name.

## Input Arguments

### **name — Custom element name**

character vector

Name of an element supported by the type of document to which this custom element is appended. For example, specify 'div' for a custom HTML div element or 'w:p' for a custom Word paragraph element.

## Output Arguments

### **customElementObj — Custom element**

mlreportgen.dom.CustomElement object

Custom element, represented by an mlreportgen.dom.CustomElement object.

## Properties

### **CustomAttributes — Custom attributes of document element**

array of mlreportgen.dom.CustomAttribute objects

Custom attributes of this document element, specified as an array of mlreportgen.dom.CustomAttribute objects. The custom attributes must be supported by the output format.

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **Name — Element name**

character vector

Element name, specified as a character vector.

### Style — Format specification

array of format objects

This property is ignored.

### StyleName — Name of custom element style

character vector

This property is ignored.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Methods

Method	Purpose
append	Append a custom element to the document element
clone	Copy custom element.
Use CustomElement.clone similar to how you use Paragraph.clone.	

## Examples

### Create a Check Box Custom Element

This example shows how to add a custom element that provides a check box in an HTML report.

Create and a custom element and append text to it.

```
import mlreportgen.dom.*;
d = Document('test');

input1 = CustomElement('input');
input1.CustomAttributes = {
    CustomAttribute('type', 'checkbox'), ...
    CustomAttribute('name', 'vehicle'), ...
    CustomAttribute('value', 'Bike'), ...
};
append(input1, Text('I have a bike'));
```

Append the custom element to an ordered list and display the report.

```

ol = OrderedList({input1});
append(d,ol);

close(d);
rptview(d.OutputPath);

```

### Create a Check Box in a Microsoft® Word Document

This example uses `mlreportgen.dom.CustomElement` and `mlreportgen.dom.CustomAttribute` objects to generate Open Office XML (OOXML) markup that displays a check box control in a Word document. For more information, see the [OOXML documentation](#) on the Office Open XML website.

Import the DOM API package.

```
import mlreportgen.dom.*;
```

Use objects of the `mlreportgen.dom.CustomElement` class to create Structured Document Tag (SDT) block-level containers for the check box control and an SDT properties element.

```
cbBlock = CustomElement('w:sdt');
cbBlockProps = CustomElement('w:stdPr');
```

The `initState` and `initStateChar` variables set the initial state of the check box. In this example we set the initial state of the check box to "unchecked" by setting `initState='0'` and `initStateChar='☐'`. If you want the initial state of the check box to be "checked", set `initState='1'` and `initStateChar='☑'`.

```
initState = '0';
initStateChar = '☐';
```

Create a check box control element and a check box state element, then append the check box state element to the check box control element.

```
cbControl = CustomElement('w14:checkbox');
cbState = CustomElement('w14:checked');
cbState.CustomAttributes = {CustomAttribute('w14:val',initState)};
append(cbControl,cbState);
```

Specify the font family and character to render a checked check box.

```
cbCheckedState = CustomElement('w14:checkedState');
cbCheckedState.CustomAttributes = { ...
    CustomAttribute('w14:val','2612'),...
    CustomAttribute('w14:font','MS Gothic')};
append(cbControl,cbCheckedState);
```

Specify the font family and character to render an unchecked check box.

```
cbUncheckedState = CustomElement('w14:uncheckedState');
cbUncheckedState.CustomAttributes = { ...
    CustomAttribute('w14:val','2610'),...
    CustomAttribute('w14:font','MS Gothic')};
append(cbControl,cbUncheckedState);
```

Append the check box control to the SDT properties element.

```
append(cbBlockProps,cbControl);
```

Append the check box control property to the SDT element.

```
append(cbBlock,cbBlockProps);
```

Append an element to indicate the end of the properties section of the SDT element.

```
append(cbBlock,CustomElement('w:stdEndPr'));
```

Create a block-level container to specify the initial state and character of the check box, then append the check box element to the container.

```
cbBlockContent = CustomElement('w:stdContent');  
textRange = CustomElement('w:r'); % text-block element  
append(textRange,Text(initStateChar));  
append(cbBlockContent,textRange);  
append(cbBlock,cbBlockContent);
```

Create an `mlreportgen.dom.Document` object, then append a title to the document object.

```
wordDoc = Document('worddoc-w-checkbox','docx');  
docTitle = Text(...  
    'Using CustomElement objects to create a check box in a Microsoft® Word Document',...  
    'Title');  
docTitle.FontSize = '12pt';  
append(wordDoc,docTitle);
```

Create an `mlreportgen.dom.Paragraph` object, then append the check box element to the paragraph object.

```
para = Paragraph();  
append(para,cbBlock);
```

Append text to the paragraph object, then add the paragraph object to the document object.

```
checkBoxStr = Text(' This is my check box');  
checkBoxStr.WhiteSpace = 'pre'; % Preserve the white spaces  
append(para,checkBoxStr);  
append(wordDoc,para);
```

Close the document object to generate the report, then open the report.

```
close(wordDoc);  
rptview(wordDoc);
```

## See Also

`mlreportgen.dom.CustomAttribute` | `mlreportgen.dom.CustomText`



# mlreportgen.dom.CustomText class

**Package:** mlreportgen.dom

Plain text appended to custom element

## Description

Plain text to append to a custom element.

## Construction

`customTextObj = CustomText()` creates an empty CustomText object.

`customTextObj = CustomText(text)` creates a CustomText object containing the specified text.

## Input Arguments

**text — Text to append to custom element**

character vector

Text to append to custom element, specified as a character vector.

## Output Arguments

**customTextObj — Text to append to custom element**

mlreportgen.dom.CustomText object

Text to append to a custom element, returned as an mlreportgen.dom.CustomText object.

## Properties

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Value — Text to add**

character vector

Text to add to a custom element, specified as a character vector.

**Examples****Create Custom Text for a Script**

```
import mlreportgen.dom.*;
d = Document('test');

script = CustomElement('script');
append(script, CustomText('document.write("Hello World!")'));
append(d, script);

close(d);
rptview('test', 'html');
```

**See Also**

[mlreportgen.dom.CustomElement](#) | [mlreportgen.dom.CustomAttribute](#)

# mlreportgen.dom.DebugMessage class

**Package:** mlreportgen.dom

Debugging message

## Description

Creates debugging message text originating from the specified source object.

## Construction

`debugMsgObj = DebugMessage(text, sourceObject)` creates a debugging message with the specified text, originating from the specified source object.

### Input Arguments

**text — Message text**

character vector

The text to display for the message.

**sourceObject — DOM object from which message originates**

a DOM object

The DOM object from which the message originates.

### Output Arguments

**debugMsgObj — Debugging message**

mlreportgen.dom.DebugMessage object

Debug message, represented by an mlreportgen.dom.DebugMessage object.

## Properties

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Source — Source object message originates from**

a DOM object

Source DOM object from which the message originates.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Text — Text of the message

character vector

Message text, specified as a character vector.

## Methods

Use DebugMessage methods similar to how you use ProgressMessage methods.

Method	Purpose
formatAsHTML	Format message as HTML.
formatAsText	Format message as text.
passesFilter	Determine whether message passes filter.

## Examples

### Create a Debug Message

Create the report document.

```
import mlreportgen.dom.*;
d = Document("test","html");
```

Create the listener and add it to the message dispatcher.

```
dispatcher = MessageDispatcher.getTheDispatcher;
dispatcher.Filter.DebugMessagesPass = true;

l = addlistener(dispatcher,"Message", ...
    @(src, evtdata) disp(evtdata.Message.formatAsText));
```

Add report content.

```
open(d);

p = Paragraph("Chapter ");
p.Tag = "chapter title";
p.Style = {CounterInc("chapter"),...
    CounterReset("table"),WhiteSpace("pre") };
append(p,AutoNumber("chapter"));
append(d,p);
```

Generate the report.

```
close(d);
rptview(d);
```

Delete the listener to avoid duplicate reporting of message objects during a MATLAB session.

`delete(l);`

**See Also**

`dispatch`

**Topics**

“Display Progress and Debugger Messages” on page 13-115

## mlreportgen.dom.Display class

**Package:** mlreportgen.dom

Display option for DOM objects

### Description

For Microsoft Word reports, specifies whether to display an `mlreportgen.dom.Text` object. For HTML reports, specifies how to display DOM objects such as text, paragraphs, images, and list items.

### Construction

`disp = Display()` in an HTML report displays a DOM object as an inline element. Word reports ignore `mlreportgen.dom.Display` objects that you create with this syntax.

`disp = Display(value)` applies the specified display value to the DOM object. For Word reports, the display option you can use is `none` and the only DOM object it applies to is a `Text` object.

### Input Arguments

#### value — Display option

character vector

Display option, specified as a character vector. The default option is `inline`.

For Microsoft Word and PDF reports, the only supported option is `none`.

Value	Display of Text or Paragraph Object
'inline'	Inline element (similar to an HTML <code>&lt;span&gt;</code> element). (Default)
'block'	Block element (similar to an HTML <code>&lt;span&gt;</code> element).
'flex'	Block-level flex container.
'initial'	Uses the default value of <code>inline</code> .
'inline-block'	Inline-level block container. Displays the inside of the block as a block-level box, and formats the object itself as an inline-level box.
'inline-flex'	Inline-level flex container.
'inline-table'	Inline-level table.
'list-item'	Similar to an HTML <code>&lt;li&gt;</code> bulleted list element.
'none'	Not displayed (has no effect on layout).  This is the only display option that applies to Word and PDF reports. In Word, if you enable <b>File &gt; Options &gt; Display &gt; Hidden text</b> , the text displays in the report.
'run-in'	As <code>block</code> or <code>inline</code> , depending on the context. For example, if the object is inside a block, the object displays as a block.
'table'	Similar to an HTML <code>&lt;table&gt;</code> element.

Value	Display of Text or Paragraph Object
'table-caption'	Similar to an HTML <caption> element.
'table-cell'	Similar to an HTML <td> element.
'table-column'	Similar to an HTML <col> element.
'table-column-group'	Similar to an HTML <colgroup> element.
'table-footer-group'	Similar to an HTML <tfoot> element.
'table-header-group'	Similar to an HTML <thead> element.
'table-row'	Similar to an HTML <tr> element.
'table-row-group'	Similar to an HTML <tbody> element.

---

**Note** The `Display` class does not support the CSS `display` value of `inherit`.

---

For details about the CSS `display` property, see [https://www.w3schools.com/cssref/pr\\_class\\_display.asp](https://www.w3schools.com/cssref/pr_class_display.asp).

## Properties

### Id — ID for Display object

character vector

A session-unique ID is generated as part of `Display` object creation. You can specify an ID to replace the generated ID.

### Tag — Tag for Display object

character vector

Tag for `Display` object, specified as a character vector.

A session-unique ID is generated as part of HTML object creation. The generated tag has the form `CLASS:ID`, where `CLASS` is the class of the element and `ID` is the value of the `Id` property of the object. You can specify a tag to replace the generated tag.

Specify your own tag value, for example, to make it easier to identify where an issue occurred during presentation generation.

### Value — Display option

character vector

Display option, specified as a character vector. For a list of options, see the description of the `value` constructor input argument.

## Examples

### **Hide Text in a Paragraph**

In Word, make sure the **File > Options > Display > Hidden text** option is cleared. This is the default setting.

```
import mlreportgen.dom.*;
rpt = Document('MyDispRep', 'docx');

t1 = Text('Hello');
t1.Style = {Display('none')};

p1 = Paragraph();
append(p1, t1);
t2 = Text('World');
append(p1, t2);
append(rpt, p1);

close(rpt);
rptview('MyDispRep', 'docx');
```

### **See Also**

`mlreportgen.dom.Text`

### **Topics**

“Report Formatting Approaches” on page 13-17

### **External Websites**

[https://www.w3schools.com/cssref/pr\\_class\\_display.asp](https://www.w3schools.com/cssref/pr_class_display.asp)  
[www.w3schools.com/tags](http://www.w3schools.com/tags)  
[www.w3schools.com/cssref](http://www.w3schools.com/cssref)

### **Introduced in R2015b**



# mlreportgen.dom.Document class

**Package:** mlreportgen.dom

Document container

## Description

Use an object of the mlreportgen.dom.Document class to represent a Document Object Model (DOM) document. Use Document object properties to specify:

- Whether to generate an HTML, Microsoft Word, or PDF document
- Where and how to store the generated document
- The template to use to format the document

The mlreportgen.dom.Document class is a handle class.

## Class Attributes

ConstructOnLoad	true
HandleCompatible	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

documentObj = mlreportgen.dom.Document() creates a Document object with default property values, which specify an output file named Untitled.htmx in the current folder, using the default HTML template.

documentObj = mlreportgen.dom.Document(outputPath) specifies the path and name of the output file and sets the OutputPath property to outputPath.

documentObj = mlreportgen.dom.Document(outputPath, type) also specifies the output type and sets the Type property to type.

documentObj = mlreportgen.dom.Document(outputPath, type, templatePath) also specifies the path and name of the template file and sets the TemplatePath property to templatePath.

## Properties

### OutputPath — Path and name of output file or folder for this document

string scalar | character vector

Path and name of the output file or folder for this document, specified as a string scalar or character vector. You can specify an absolute path or a path relative to the current folder. You can set this property only before opening the document.

The default value is the path of a file or folder named `untitled` in the current folder. If `OutputPath` specifies a file and you do not specify the file extension, the DOM API adds an extension based on the `Type` property.

Whether `OutputPath` specifies the path of a file or folder depends on the value of the `PackageType` property, as shown in the table.

PackageType	OutputPath Value
"zipped" or "single-file"	Path and name of ZIP file or single file
"unzipped"	Folder for the unzipped files
"both"	Path and name of ZIP file and folder for the unzipped files

**Note** Generating a PDF report on a cloud drive, such as MATLAB Drive™, can result in an error that is caused by file contention between the report generation software and the cloud drive synchronization software. To avoid this error, generate reports on a local drive that does not synchronize with the cloud. Consider writing a script that generates a report on a local drive and then copies the report to the cloud drive.

#### Attributes:

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>
<code>NonCopyable</code>	<code>true</code>

#### PackageType — Packaging for generated files

"zipped" | "unzipped" | "both" | "single-file"

Packaging used for the generated files, specified as one of the values in the table. You can specify `PackageType` as a string scalar or character vector.

Value	Supported Report Types	Description
"zipped"	"docx" or "html"	Generates the report as a ZIP file at the location specified by the <code>OutputPath</code> property. The ZIP file has an extension that matches the document type (.docx for Word output or .html for HTML output.) For example, if the document <code>Type</code> is <code>docx</code> and <code>OutputPath</code> is <code>s:\docs\MyDoc</code> , the output is packaged in a ZIP file named <code>s:\docs\MyDoc.docx</code> .

Value	Supported Report Types	Description
"unzipped"	"docx" or "html"	Generates the report as separate files in a folder that has the file name of the OutputPath property. For example, if the OutputPath is s:\docs\MyDoc, the output folder is s:\docs\MyDoc.
"both"	"docx" or "html"	Generates zipped and unzipped outputs.
"single-file"	"pdf" or "html-file"	Generates the report as a single file.

To generate an HTML report that you can open without unzipping, set `Type` to "html" and `PackageType` to "unzipped" or "both". In the folder that contains the generated files, open the `root.html` file.

**Attributes:**

```
GetAccess                public
SetAccess                public
NonCopyable              true
```

**ForceOverwrite — Whether to overwrite existing output file**

true (default) | false

Whether to overwrite an existing output file, specified as true or false. Set this property to true to overwrite an existing output file with the same name. If this property is false and a writable file with the same name exists, closing this document causes an error. If the existing file is read-only, closing this document causes an error regardless of this property setting.

**Attributes:**

```
GetAccess                public
SetAccess                public
NonCopyable              true
```

**StreamOutput — Option to stream output to disk**

false (default) | true

Option to stream the output to disk, specified as true or false. By default, document elements are stored in memory until the document is closed. Set this property to true to write the document elements to disk as the elements are appended to the document.

**Attributes:**

```
GetAccess                public
SetAccess                public
NonCopyable              true
```

**TitleBarText — Text for HTML browser title bar**

string scalar | character vector

Text for the HTML browser title bar, specified as a string scalar or character vector. For HTML documents, this property specifies the text that appears in the title bar of the browser used to display this document. Word and PDF documents ignore this property.

Set this property before opening the document.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**HTMLHeadExt — Custom content for HTML header**

string scalar | character vector

Custom content for the HTML header, specified as a string scalar or character vector. The value of this property is appended to the <head> element of this document after the content specified by the head section of the document template. Set this property only before opening the document.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**TemplatePath — Path of template to use**

string scalar | character vector

Full path of the template to use, specified as a string scalar or character vector.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Type — Type of output**

"html" (default) | "docx" | "pdf" | "html-file"

Type of output, specified as one of these string scalars or character vectors:

- "html" — HTML output packaged as a zipped or unzipped folder that contains the HTML document text, image, style sheet, and JavaScript files
- "docx" — Word output
- "pdf" — PDF output
- "html-file" — A single HTML file that contains the text, style sheets, JavaScript, and images for the report

If you specify a template using the `TemplatePath` property, the template must be consistent with the `Type` argument. You must specify a template with a Word template extension (`.dotx`) for Word output, an HTML template package extension (`.htmxtx`) for HTML output, a PDF template package extension (`.pdftx`) for PDF output, and a single-file HTML template extension (`.html`) for `html-file` output.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**CurrentHoleId — ID of current hole in document**

character vector

ID of the current hole in the document, specified as a character vector.

**Attributes:**

GetAccess	public
SetAccess	private
Transient	true
NonCopyable	true

**CurrentHoleType — Type of current hole**

'Inline' | 'Block'

Type of the current template hole, specified as 'Inline' or 'Block'.

- An inline hole is for document elements that a paragraph element can contain: Text, Image, LinkTarget, ExternalLink, InternalLink, CharEntity, or AutoNumber.
- A block hole can contain a Paragraph, Table, OrderedList, UnorderedList, DocumentPart, or Group element.

**Attributes:**

GetAccess	public
SetAccess	private
Transient	true
NonCopyable	true

**CurrentPageLayout — Current page layout of this document**

mlreportgen.dom.DOCXPageLayout object | mlreportgen.dom.PDFPageLayout object | []

Current page layout of this document, specified as an mlreportgen.dom.DOCXPageLayout object, mlreportgen.dom.PDFPageLayout object, or []. This property applies to Word and PDF documents. For Word documents, the value is a DOCXPageLayout object that specifies the current page layout. For PDF documents, the value is a PDFPageLayout object if the document currently specifies a page layout. For HTML documents, the value is always [].

**Attributes:**

GetAccess	public
SetAccess	private
Transient	true
NonCopyable	true

**OpenStatus — Open status of document**

'unopened' (default) | 'open' | 'closed'

Open status of this document, specified as 'unopened', 'open', or 'closed'.

**Attributes:**

GetAccess	public
SetAccess	private
Transient	true
NonCopyable	true

**Parent — Parent of this DOM API object**

DOM API object

Parent of this DOM API object, specified as a DOM API object.

**Attributes:**

GetAccess	public
SetAccess	private
Transient	true
NonCopyable	true

**Children — Children of this DOM API object**

array of DOM API objects

Children of this DOM API object, specified as an array of DOM API objects.

**Attributes:**

GetAccess	public
SetAccess	private
Transient	true
NonCopyable	true

**Tag — Tag for this DOM API object**

character vector | string scalar

Tag for this DOM API object, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Id — ID for this DOM API object**

character vector | string scalar

ID for this DOM API object, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Methods****Public Methods**

addHTML	Append HTML string to document
addHTMLFile	Append HTML file contents to document
append	Append DOM or MATLAB object to document
close	Close document
createAutoNumberStream	Create numbering stream
mlreportgen.dom.Document.createTemplate	Create DOM template file
fill	Fill document holes with generated content
getAutoNumberStream	Return numbering stream
mlreportgen.dom.Document.getCoreProperties	Get document or template core properties
mlreportgen.dom.Document.getImageDirectory	Get image folder of document
mlreportgen.dom.Document.getImagePrefix	Get generated image name prefix
getMainPartPath	Return path of main part of document output package
mlreportgen.dom.Document.getOPCMainPart	Return main part of document, document part, or template
moveToNextHole	Move document append point to next template hole
open	Open document
package	Add OPC part files to document package
mlreportgen.dom.Document.setCoreProperties	Set OPC core properties of output document or template

**Examples****Create a Word Document**

Create a Word document, add content, and view the report in Word.

```
import mlreportgen.dom.*;
d = Document("mydoc", "docx");

append(d, "Hello World");

close(d);
rptview(d);
```

**Create an HTML Document as a Single File**

Create an HTML document as a single HTML file that includes an image. The example assumes that there is a MyImage.jpg file and a myHTMLTemplate.html HTML template file.

Create a document whose output is a single HTML file and uses the template `myHTMLTemplate`. Add text and an image to the report. Close and view the document.

```
import mlreportgen.dom.*;
d = Document("mydoc", "html-file", "myHTMLTemplate.html");
open(d);

append(d, "Hello world");
append(d, Image("C:/images/LocalSystem/MyImage.jpg"));

close(d);
rptview(d);
```

### **See Also**

`mlreportgen.dom.DocumentPart` | `mlreportgen.dom.Text` | `mlreportgen.dom.Paragraph` | `mlreportgen.dom.Image`

### **Topics**

“Create Report Container” on page 13-8

**Introduced in R2014b**



# mlreportgen.dom.DocumentPart class

**Package:** mlreportgen.dom

Create a document part object

## Description

Define a document part, a repeatable part of a report. A document part typically has holes that you fill during report generation. You can append a part to a document or to a document part of the same output type.

## Construction

`documentPartObj = DocumentPart()` creates an HTML document part using the default HTML template.

`documentPart = DocumentPart(type)` creates a document part of the specified type (for example, Microsoft Word) based on the default template for that part.

`documentPartObj = DocumentPart(type, templatePath)` creates a document part based on the specified template.

`documentPartObj = DocumentPart(type, templatePath, docPartTemplateName)` creates a document part based on the specified document part template in the specified template.

`documentPartObj = DocumentPart(templateSrc, docPartTemplateName)` creates a document part based on the specified document part template stored in the template used by the specified source. The source can be a document or a document part.

## Input Arguments

### type — Type of output

'html' (default) | 'docx' | 'pdf' | 'html-file'

Type of output, specified as one of these values:

- 'html' — HTML output
- 'pdf' — PDF based on a PDF template
- 'docx' — Word output
- 'html-file' — HTML output, using a single file that contains the CSS, JavaScript, and images for the report

If you specify a template using the `templatePath` argument, the value for `type` must match the template type.

### templatePath — Path of this part's template

[] (default) | character vector

Full path of this part's template file or folder, specified as a character vector. If you omit a file extension, the template type is based on the document type, for example, `.docx` for Word.

Data Types: char

**docPartTemplateName — Document part template name**

character vector

Document part template name, specified as a character vector. Specify where the part is stored using the `templatePath` or `templateSrc` argument.

**templateSrc — Document or document part that holds the document part template**

`microsoftreportgen.dom.Document` object | `microsoftreportgen.dom.DocumentPart` object

Document or document part object whose template contains the template for this document part, specified as an `microsoftreportgen.dom.Document` object for a document or an `microsoftreportgen.dom.DocumentPart` object for a document part.

**Output Arguments****documentPartObj — Document part**

`microsoftreportgen.dom.DocumentPart` object

Document part, returned as an `microsoftreportgen.dom.DocumentPart` object.

**Properties****Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**CurrentHoleId — ID of current hole in document**

character vector

This read-only property is the hole ID of the current hole in this document.

**CurrentHoleType — Type of current hole**

'Inline' | 'Block'

Type of the current template hole, specified as 'Inline' or 'Block'.

- An inline hole is for document elements that a paragraph element can contain: Text, Image, LinkTarget, ExternalLink, InternalLink, CharEntity, AutoNumber.
- A block hole can contain a Paragraph, Table, OrderedList, UnorderedList, DocumentPart, or Group.

**CurrentPageLayout — Current page layout of this document**

`microsoftreportgen.dom.DOCXPageLayout` object | `microsoftreportgen.dom.PDFPageLayout` object

This property applies to Word and PDF documents. For Word documents, the value is a `DOCXPageLayout` object that specifies the current page layout. For PDF documents, the value is a `PDFPageLayout` object if the document currently specifies a page layout. For HTML documents, the value is always [].

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **OpenStatus — Open status of document element**

unopened (default) | open | closed

This read-only property lists the open status of this document element.

### **Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### **TemplateName — Name of this part's template**

character vector

The name of this part's template if the template is stored in the document part template library of the template specified by this part's TemplatePath property. If this property is [], the template specified by the TemplatePath property is used as this part's template.

### **TemplatePath — Path of the template**

character vector

Path of this part's template or of a template whose template library contains this part's template, specified as a character vector.

### **Type — Output type**

'html' | 'html-file' | 'docx' | 'pdf'

Output type, specified as one of these values.

- 'HTML' - HTML report packaged as a zipped file containing the HTML file, images, style sheet, and JavaScript files of the report.
- 'HTML-FILE' - HTML report as a single HTML file containing the text, style sheet, JavaScript, and base64-encoded images of the report
- 'PDF' - PDF file
- 'DOCX' - Microsoft Word document

If you specify a template using the TemplatePath property, the value for Type must match the template type.

## **Methods**

Use DocumentPart methods like you use the corresponding Document methods.

Method	Purpose
addHTML Use <code>DocumentPart.addHTML</code> in a similar way to how you use <code>Document.addHTML</code> .	Append HTML text to document
addHTMLFile Use <code>DocumentPart.addHTMLFile</code> in a similar way to how you use <code>Document.addHTMLFile</code> .	Append HTML file contents to document
append	Append document element to the document part.
close	Close this document part. You cannot close a document part if it has not been opened or was previously closed.
<code>mlreportgen.dom.Document.createTemplate</code>	Create document part template.
fill	Fill document hole.
<code>mlreportgen.dom.Document.getCoreProperties</code>	Get core properties of document part.
<code>mlreportgen.dom.Document.getOPCMainPart</code>	Get full path of main part of output document.
moveToNextHole	Move to next template hole.
open	Open this document part. You cannot open a document part if it was previously opened or closed. You also cannot open a document part if its library source is closed.
<code>mlreportgen.dom.Document.setCoreProperties</code>	Set core properties of document part.

## Examples

### Document Part from Blank Document Part Template

This example creates a function `createMagicParts` that defines a document part based on a blank document part template. The new document part has a heading whose text depends on the input. Each document part generated contains a magic square table whose appearance is also based on the input. The example creates a containing function `magicparts` that appends the document part to the report iteratively based on the input.

Create the function.

```
function magic_square_report(square_sizes, report_type)
%MAGIC_SQUARE_REPORT Report on magic squares
%   magic_square_report(square_sizes, report_type)
%   creates a report of the specified output type
%   (docx, pdf, or html) on the specified magic
%   squares. For example, to create a PDF report on
%   squares of size 5, 10, and 15, enter the following
%   line at the MATLAB command line:
```

```

%
%      magic_square_report([5,10,15], 'pdf');

import mlreportgen.dom.*;
rpt = Document('MagicSquareReport', report_type);
open(rpt);
for i = 1:length(square_sizes)
    sz = square_sizes(i);
    section = createSquareSection(rpt,sz);
    append(rpt,section);
end
close(rpt);
rptview(rpt.OutputPath);

function section = createSquareSection(rpt,square_size)
import mlreportgen.dom.*;
% Create document part to hold section
section = DocumentPart(rpt.Type);
% Create magic square heading
h1 = Heading1(sprintf('magic(%i)',square_size));
% Put each square on a separate page.
h1.Style = {PageBreakBefore(true)};
append(section,h1);
% Create table to hold square
table = append(section, Table(magic(square_size)));
% Format table
table.Border = 'solid';
table.ColSep = 'solid';
table.RowSep = 'solid';

```

Call the function to generate the report. Change the input arguments to change the contents or output format. This example creates a Word document that contains three squares.

```
magic_square_report([5,8,12], 'docx');
```

## See Also

mlreportgen.dom.Document

## Topics

“Use Subforms in a Report” on page 13-26

“Form-Based Reporting” on page 13-24

## mlreportgen.dom.DOCXPageFooter class

**Package:** mlreportgen.dom

Page footer definition for Microsoft Word document

### Description

Add a footer to the first page of a Word document layout or to odd pages, even pages, or both.

### Construction

`docxFooter = DOCXPageFooter()` creates a page footer based on the default Word template.

`docxFooter = DOCXPageFooter(pageType)` creates a page footer for the specified type of page, that is, odd, even, or first, based on the default Word template.

`docxFooter = DOCXPageFooter(pageType, templatePath)` creates a page footer for the specified type of page based on the specified template.

`docxFooter = DOCXPageFooter(pageType, templatePath, docPartTemplateName)` creates a page footer for the specified type of page, based on the specified document part template in the specified template.

`docxFooter = DOCXPageFooter(pageType, templateSrc, docPartTemplateName)` creates a page footer for the specified type of page, based on the specified document part template from the specified source. The source can be a document or a document part.

### Input Arguments

#### **pageType — Type of pages the footer appears on**

`[]` (default) | `default` | `first` | `even`

Type of pages the footer appears on, specified as one of these values:

- `default` — Footer for odd pages of the section, even pages if you do not specify an even-page footer, and first page if you do not specify a first-page footer.
- `first` — Footer for first page of a section.
- `even` — Footer for even pages of a section.

For example, to make different footers appear on odd pages and on even pages, define two footers. Set `pageType` to `default` for one and to `even` for the other.

#### **templatePath — Full path of footer template**

character vector

Full path of footer template, specified as a character vector.

#### **docPartTemplateName — Document part template name**

character vector

Name of this part's template if it is stored in a template specified by the `templatePath` or `templateSrc` argument, specified as a character vector.

**templateSrc — Document or document part that holds the document part template**

`mlreportgen.dom.Document` object | `mlreportgen.dom.DocumentPart` object

Document or document part object whose template contains the template for this document part, specified as an `mlreportgen.dom.Document` object for a document or an `mlreportgen.dom.DocumentPart` object for a document part.

**Output Arguments**

**docxFooter — Page footer for Word document**

`mlreportgen.dom.DOCXPageFooter` object

Page footer for a Word document, returned as an `mlreportgen.dom.DOCXPageFooter` object.

**Properties**

**Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**CurrentPageLayout — Ignored by page footers**

not applicable

This property does not apply to page footers.

**CurrentHoleId — ID of current hole in document**

character vector

This read-only property is the hole ID of the current hole in this document.

**CurrentHoleType — Type of current hole**

'Inline' | 'Block'

Type of the current template hole, specified as 'Inline' or 'Block'.

- An inline hole is for document elements that a paragraph element can contain: Text, Image, LinkTarget, ExternalLink, InternalLink, CharEntity, AutoNumber.
- A block hole can contain a Paragraph, Table, OrderedList, UnorderedList, DocumentPart, or Group.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**PageType — Type of pages on which footer appears**

[] (default) | default | first | even

Type of page on which the footer appears, specified as one of these values:

- `default` — Footer for odd pages of the section, even pages if you do not specify an even-page footer, and first page if you do not specify a first-page footer.
- `first` — Footer for first page of a section.
- `even` — Footer for even pages in a section.

To have a footer appear on odd pages and on even pages, define two footers, one with `pageType` set to `default` and the other with `pageType` set to `even`.

### Parent — Parent of document element

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### TemplatePath — Path to template used for footer

character vector

Full path to the template to use for this footer, specified as a character vector.

## Methods

Use `DocumentPageFooter` methods as you use the corresponding `Document` methods.

Method	Purpose
<code>append</code>	Append one of these DOM objects to the footer: <ul style="list-style-type: none"> <li>• <code>CustomElement</code></li> <li>• <code>FormalTable</code></li> <li>• <code>Group</code></li> <li>• <code>ExternalLink</code></li> <li>• <code>Image</code></li> <li>• <code>InternalLink</code></li> <li>• <code>OrderedList</code></li> <li>• <code>Paragraph</code></li> <li>• <code>RawText</code></li> <li>• <code>Table</code></li> <li>• <code>Text</code></li> <li>• <code>UnorderedList</code></li> </ul>
<code>close</code>	Close the footer.



Method	Purpose
fill	Fill the template hole.
moveToNextHole	Move to the next template hole.
open	Open the footer.

## Examples

### Add Footer to Word Document

This example defines first, even, and odd page footers in a Word document. It inserts a page number in each footer, using a different alignment for each page type.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'docx');
open(d);

% Create page footer objects for each type of page
% Assign a matrix of page footer objects to the current page layout
firstfooter = DOCXPageFooter('first');
evenfooter = DOCXPageFooter('even');
oddfooter = DOCXPageFooter('default');
d.CurrentPageLayout.PageFooters = [firstfooter, evenfooter, oddfooter];

% Add title to first page footer
p = Paragraph('My Document Title');
p.HAlign = 'center';
append(d.CurrentPageLayout.PageFooters(1), p);

% Add page number to even page footer
% Align even page numbers left
pg2 = Page();
p2 = Paragraph();
p2.HAlign = 'left';
append(p2, pg2);
append(d.CurrentPageLayout.PageFooters(2), p2);

% Add page number to odd page footer
% Align odd page numbers right
pg3 = Page();
p3 = Paragraph();
p3.HAlign = 'right';
append(p3, pg3);
append(d.CurrentPageLayout.PageFooters(3), p3);

% Create several pages.
p = Paragraph('Hello World');
append(d, p);
p = Paragraph('Another page');
p.Style = {PageBreakBefore(true)};
append(d, p);
append(d, clone(p));
```

```
close(d);  
rptview(d.OutputPath);
```

### **See Also**

[mlreportgen.dom.DOCXPageHeader](#) | [mlreportgen.dom.DOCXPageLayout](#) |  
[mlreportgen.dom.PDFPageHeader](#) | [mlreportgen.dom.PDFPageFooter](#) |  
[mlreportgen.dom.PDFPageLayout](#)

### **Topics**

“Create Page Footers and Headers” on page 13-148

# mlreportgen.dom.DOCXPageHeader class

**Package:** mlreportgen.dom

Page header definition for Microsoft Word document

## Description

Add a header to the first page of a section or to odd pages, even pages, or both.

## Construction

`docxHeader = DOCXPageHeader()` creates a page header based on the default Word template.

`docxHeader = DOCXPageHeader(pageType)` creates a page header for the specified type of page, that is, odd, even, or first, based on the default Word template.

`docxHeader = DOCXPageHeader(pageType, templatePath)` creates a page header for the specified type of page based on the specified template.

`docxHeader = DOCXPageHeader(pageType, templatePath, docPartTemplateName)` creates a page header for the specified type of page, based on the specified document part template in the specified template.

`docxHeader = DOCXPageHeader(pageType, templateSrc, docPartTemplateName)` creates a page header for the specified type of page, based on the specified document part template used by the specified source. The source can be a document or a document part.

## Input Arguments

### **pageType** — Type of pages header appears on

`[] (default) | default | first | even`

Type of page header appears on, specified as one of these values:

- `default` — Header for odd pages of the section, even pages if you do not specify an even-page header, and first page if you do not specify a first-page header.
- `first` — Header for first page of a section.
- `even` — Header for even pages in a section.

For example, to have a blank header appear on the first page of a section and a different header appear on the other pages, define two headers, one with `pageType` set to `first` and the other with `pageType` set to `default`.

### **templatePath** — Full path of header template

character vector

Full path of header template, specified as a character vector.

### **docPartTemplateName** — Document part template name

character vector

Name of this part's template if it is stored in a template specified by the `templatePath` or `templateSrc` argument, specified as a character vector.

**templateSrc — Document or document part that holds the document part template**

`mreportgen.dom.Document` object | `mreportgen.dom.DocumentPart` object

Document or document part object whose template contains the template for this document part, specified as an `mreportgen.dom.Document` object for a document or an `mreportgen.dom.DocumentPart` object for a document part.

**Output Arguments****docxHeader — Page header for Word document**

`mreportgen.dom.DOCXPageHeader` object

Page header for a Word document, returned as an `mreportgen.dom.DOCXPageHeader` object.

**Properties****Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**CurrentHoleId — ID of current hole in document**

character vector

This read-only property is the hole ID of the current hole in this document.

**CurrentHoleType — Type of current hole**

'Inline' | 'Block'

Type of the current template hole, specified as 'Inline' or 'Block'.

- An inline hole is for document elements that a paragraph element can contain: Text, Image, LinkTarget, ExternalLink, InternalLink, CharEntity, AutoNumber.
- A block hole can contain a Paragraph, Table, OrderedList, UnorderedList, DocumentPart, or Group.

**CurrentPageLayout — Current page layout of this document**

`mreportgen.dom.DOCXPageLayout` object | `mreportgen.dom.PDFPageLayout` object

This property applies to Word and PDF documents. For Word documents, the value is a `DOCXPageLayout` object that specifies the current page layout. For PDF documents, the value is a `PDFPageLayout` object if the document currently specifies a page layout. For HTML documents, the value is always [].

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**PageType — Type of pages header appears on**

[] (default) | default | first | even

Type of page header appears on, specified as one of these values:

- `default` — Header for odd pages of the section, even pages if you do not specify an even-page header, and first page if you do not specify a first-page header.
- `first` — Header for first page of a section.
- `even` — Header for even pages in a section.

For example, to have a blank header appear on the first page and a different header appear on the other pages, define two headers, one with `pageType` set to `first` and the other with `pageType` set to `default`.

#### **Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

#### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

#### **TemplatePath — The path to template used for header**

character vector

Full path to the template to use for this header, specified as a character vector.

## **Methods**

Use `DOCXPageHeader` methods as you use the corresponding `Document` methods.

Method	Purpose
append	Append one of these DOM objects to the header: <ul style="list-style-type: none"><li>• CustomElement</li><li>• DOCXPageLayout</li><li>• FormalTable</li><li>• Group</li><li>• ExternalLink</li><li>• Image</li><li>• InternalLink</li><li>• OrderedList</li><li>• Paragraph</li><li>• RawText</li><li>• Table</li><li>• Text</li><li>• UnorderedList</li></ul>
close	Close header.
fill	Fill template hole.
moveToNextHole	Move to next template hole.
open	Open header.

### See Also

`mlreportgen.dom.DOCXPageFooter` | `mlreportgen.dom.DOCXPageLayout` |  
`mlreportgen.dom.PDFPageHeader` | `mlreportgen.dom.PDFPageFooter` |  
`mlreportgen.dom.PDFPageLayout`

### Topics

“Create Page Footers and Headers” on page 13-148

# mlreportgen.dom.DOCXPageLayout class

**Package:** mlreportgen.dom mlreportgen.dom

Page format and layout for section of Microsoft Word document

## Description

Use an mlreportgen.dom.DOCXPageLayout object to define the page format, headers, and footers of a section of a Microsoft Word document.

The mlreportgen.dom.DOCXPageLayout class is a handle class.

## Class Attributes

HandleCompatible	true
ConstructOnLoad	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

pageLayoutObj = mlreportgen.dom.DOCXPageLayout() creates a DOCXPageLayout object with default property on page 12-95 values.

## Properties

### PageHeaders — Page headers for this layout

array of mlreportgen.dom.DOCXPageHeader objects

Page headers for this layout, specified as an array of mlreportgen.dom.DOCXPageHeader objects. You can define up to three page headers for a layout, one each for:

- The first page of the section
- Even pages
- Odd pages

### PageFooters — Page footers for this layout

array of mlreportgen.dom.DOCXPageFooter objects

Page footers for this layout, specified as an array of mlreportgen.dom.DOCXPageFooter objects. You can define up to three page footers for a layout, one each for:

- The first page of the section
- Even pages
- Odd pages

**FirstPageNumber — Number of first page in section**

integer

Number of the first page in a section, specified as an integer.

---

**Note** Page numbers are rendered only on `microsoftreportgen.dom.Page` objects that you append to the document or to any part of the document. To render page numbers in the page headers or footers, assign an `microsoftreportgen.dom.DOCXPageHeader` object to the `PageHeaders` property or an `microsoftreportgen.dom.DOCXPageFooter` object to the `PageFooters` property. Then append an `microsoftreportgen.dom.Page` object to the `DOCXPageHeader` or `DOCXPageFooter` object. For example, see “Add and Customize Page Numbers in a Microsoft® Word Document” on page 12-99. Alternatively, you can append `microsoftreportgen.dom.Page` objects to the document itself, or to elements in the document, but the page number renders only on the part of the document where you append the `Page` object.

---

**PageNumberFormat — Type of page numbering to use**

character vector | string scalar

Type of page numbering to use, specified as one of the character vectors or string scalars in the table.

Value	Meaning	Applies To	
		DOCX	PDF
'a'	Lowercase alphabetic	✓	✓
'A'	Uppercase alphabetic	✓	✓
'i'	Lowercase Roman numerals	✓	✓
'I'	Uppercase Roman numerals	✓	✓
'n', 'N', '1', 'decimal'	Arabic numerals	✓	✓
'numberInDash'	Number with dashes on either side	✓	
'hebrew1'	Hebrew numerals	✓	
'hebrew2'	Hebrew alphabetic	✓	
'arabicAlpha'	Arabic alphabetic	✓	
'arabicAbjad'	Arabic abjad numerals	✓	
'thaiLetters'	Thai letters	✓	
'thaiNumbers'	Thai numerals	✓	
'thaiCounting'	Thai counting system	✓	



---

**Note** Page numbers are rendered only on `mlreportgen.dom.Page` objects that you append to the document or to any part of the document. To render page numbers in the page headers or footers, assign an `mlreportgen.dom.DOCXPageHeader` object to the `PageHeaders` property or an `mlreportgen.dom.DOCXPageFooter` object to the `PageFooters` property. Then append an `mlreportgen.dom.Page` object to the `DOCXPageHeader` or `DOCXPageFooter` object. For example, see “Add and Customize Page Numbers in a Microsoft® Word Document” on page 12-99. Alternatively, you can append `mlreportgen.dom.Page` objects to the document itself, or to elements in the document, but the page number renders only on the part of the document where you append the `Page` object.

---

### **RawFormats — XML markup for unsupported layout formats**

cell array

XML markup for unsupported layout formats, specified as a cell array of character vectors. For information about XML markup for Word formats, see <https://www.ecma-international.org/publications-and-standards/standards/ecma-376/>.

### **PageMargins — Sizes of margins, header, footer, and gutter**

`mlreportgen.dom.PageMargins` object

Sizes of the margins, header, footer, and gutter for this page layout, specified as an `mlreportgen.dom.PageMargins` object.

### **PageSize — Size of pages in this layout**

`mlreportgen.dom.PageSize` object

Size of pages in this layout, specified as an `mlreportgen.dom.PageSize` object.

### **PageBorder — Page borders for this layout**

`[]` (default) | `mlreportgen.dom.PageBorder` object

Page borders for this layout, specified as an `mlreportgen.dom.PageBorder` object.

### **SectionBreak — Section break options**

'Next Page' | 'Same Page' | 'Odd Page' | 'Even Page'

Section break options, specified as one of these character vectors or string scalars:

- 'Next Page' — Start the section on the next page.
- 'Same Page' — Start the section on the same page as the current section.
- 'Odd Page' — Start the section on an odd page.
- 'Even Page' — Start the section on an even page.

### **StyleName — Ignored by page layouts**

not used

This property does not apply to page layouts.

### **Style — Formats to apply to layout**

array of format objects

Formats to apply to this layout, specified as an array of format objects. Formats that do not apply to a page layout are ignored.

**CustomAttributes — Custom attributes of document element**array of `mreportgen.dom.CustomButton` objects

Custom attributes of this document element, specified as an array of `mreportgen.dom.CustomButton` objects. The custom attributes must be supported by the output format.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Methods****Public Methods**

rotate	rotate( <code>pageLayoutObj</code> ) switches the orientation of pages between portrait and landscape. The method switches the values of the <code>Height</code> and <code>Width</code> properties of the <code>PageSize</code> object that is associated with the <code>PageSize</code> property of <code>pageLayoutObj</code> . The method also switches the value of the <code>Orientation</code> property of the <code>PageSize</code> object between 'portrait' and 'landscape'.
--------	---

**Examples**

## Change Page Margins of a Document Section

Use the `CurrentPageLayout` property of a document to access the layout object for the document. Change the left and right margins of the layout by setting the `Left` and `Right` properties of the `mlreportgen.dom.PageMargins` object used by the layout object.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'docx');

open(d);
pageLayoutObject = d.CurrentPageLayout;
pageLayoutObject.PageMargins.Left = '2in';
pageLayoutObject.PageMargins.Right = '2in';
p = Paragraph('Hello World');
append(d,p);

close(d);
rptview(d);
```

## Change the Page Orientation

By default, a `DOCXPageLayout` object specifies a page with an 11-inch height, 8.5-half inch width, and portrait orientation. To change the orientation to landscape, use the `rotate` method of the `DOCXPageLayout` object. The method updates the property values of the associated `mlreportgen.dom.PageSize` object so that the height is 8.5 inches, the width is 11 inches, and the orientation is landscape.

```
import mlreportgen.dom.*;
d = Document('myreport', 'docx');
open(d);

pageLayoutObj = d.CurrentPageLayout;
rotate(pageLayoutObj);

append(d, 'This document has landscape pages');
close(d);
rptview(d);
```

## Add and Customize Page Numbers in a Microsoft® Word Document

This example shows how to add and customize page numbers in the footer section of a generated Word document.

Import this package so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
```

Create an `mlreportgen.dom.Document` object of type DOCX with a title.

```
document = Document("my_docx_w_page_numbers_document", "docx");
heading = Heading(1, "Customize Page Numbers Example");
append(document, heading);
```

Add content to the document.

```
append(document, LineBreak);  
for pageInd = 1:5  
    append(document, sprintf("Content of page number %i", pageInd));  
    append(document, PageBreak);  
end  
append(document, Paragraph("Content of last page"));
```

Add a page footer to the document by assigning an `m1reportgen.dom.DOCXPageFooter` object to the `PageFooters` property of the current page layout. Then set the page number format of the current page layout to uppercase Roman numerals.

```
curLayout = document.CurrentPageLayout;  
curLayout.PageFooters = DOCXPageFooter();  
curLayout.PageNumberFormat = "I";
```

Append an `m1reportgen.dom.Page` object to the page footer object.

```
append(curLayout.PageFooters, Page());
```

Close and view the document.

```
close(document);  
rptview(document);
```

## See Also

`m1reportgen.dom.DocumentPart` | `m1reportgen.dom.DOCXSubDoc` |  
`m1reportgen.dom.DOCXPageFooter` | `m1reportgen.dom.DOCXPageHeader` |  
`m1reportgen.dom.PageSize` | `m1reportgen.dom.PageMargins` |  
`m1reportgen.dom.PageBorder` | `m1reportgen.dom.PageNumber` |  
`m1reportgen.dom.PageRawFormat` | `m1reportgen.dom.Page`

## Topics

“Report Formatting Approaches” on page 13-17

## Introduced in R2016a

# mlreportgen.dom.DOCXSubDoc class

**Package:** mlreportgen.dom

Reference to external Microsoft Word document

## Description

Use an object of the mlreportgen.dom.DOCXSubDoc class to create a reference to an external Microsoft Word document in a Word, PDF, or HTML document. A link to the referenced document (also known as a subdocument) is inserted in the parent document where you append the DOCXSubDoc object.

When you initially open the parent document in Word, Word displays the link to the subdocument instead of the content. To replace the link with the content, select **Expand Subdocuments** from the **Outlining** tab of the **View** tab on the Word toolstrip. The rptview command expands subdocuments when it opens a Word document. You can also use docview to expand and unlink subdocuments.

The mlreportgen.dom.DOCXSubDoc class is a handle class.

## Class Attributes

ConstructOnLoad	true
HandleCompatible	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

docxSubDocObj = mlreportgen.dom.DOCXSubDoc() creates an empty document reference.

docxSubDocObj = mlreportgen.dom.DOCXSubDoc(path) creates a reference to a Word document at the specified path and sets the Target property to path.

## Public Properties

### Id — ID for this DOM API object

character vector | string scalar

ID for this DOM API object, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Attributes:

GetAccess	public
SetAccess	public
NonCopyable	true

**Tag — Tag for this DOM API object**

character vector | string scalar

Tag for this DOM API object, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Target — Path of document targeted by this reference**

character vector | string scalar

Path of document targeted by this reference, specified as a character vector or string scalar. Use ASCII characters. Use the following format for specifying a full path involving a mapped drive.

```
'file:///C:/UserPath/FileName.docx'
```

**Attributes:**

GetAccess	public
SetAccess	immutable
Transient	true
NonCopyable	true

**Methods****Public Methods**

Method	Purpose
clone	Clone this Word document reference.
Use DOCXSubDoc.clone in a similar way to how you use Paragraph.clone.	

**Examples****Append a Word Document to a Report**

```
import mlreportgen.dom.*

info = Document('CompanyInfo', 'docx');
append(info, 'XYZ, Inc., makes widgets. ');
close(info);

infoPath = info.OutputPath;
```

```
rpt = Document('Report', 'docx');  
open(rpt);  
  
append(rpt, Paragraph('About XYZ, Inc.'));  
  
append(rpt, DOCXSubDoc(infoPath));  
  
close(rpt);  
rptview(rpt.OutputPath);
```

## Alternative Functionality

### `mlreportgen.dom.EmbeddedObject`

To insert documents other than Word documents into another document, use objects of the `mlreportgen.dom.EmbeddedObject` class.

To insert Word documents into another document, you can use objects of the `DOCXSubDoc` or `EmbeddedObject` class. To decide which class to use, consider these guidelines:

- Use `DOCXSubDoc` objects if you want the parent document to display links to subdocuments initially. You might want to display links initially when the parent document links to a set of subdocuments in an outline view. Use `EmbeddedObject` objects if you want the parent document to display the subdocument contents regardless of whether you open the parent document in Word or by using `rptview`.
- Use `EmbeddedObject` objects if you want to move the parent document without moving the subdocuments.

## See Also

`docview` | `mlreportgen.dom.DOCXSection` | `mlreportgen.dom.DocumentPart` | `mlreportgen.dom.EmbeddedObject`

### Introduced in R2014b

## mlreportgen.dom.ErrorMessage class

**Package:** mlreportgen.dom

Error message

### Description

Specifies error message text originating from a specified source object.

### Construction

`errorMsgObj = ErrorMessage(text, sourceObject)` creates an error message with the specified text originating from the specified source object.

#### Input Arguments

**text — Message text**

character vector

The text to display for the message.

**sourceObject — The DOM object from which the message originates**

a DOM object

The DOM object from which the message originates.

#### Output Arguments

**errorMsgObj — Error message**

mlreportgen.dom.ErrorMessage object

Error message, represented by an mlreportgen.dom.ErrorMessage object.

### Properties

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Source — Source object from which the message originates**

a DOM object

Source DOM object from which the message originates.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.



The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Text — Text of message

character vector

Message text, specified as a character vector.

## Methods

Use ErrorMessage methods similar to how you use ProgressMessage methods.

Method	Purpose
formatAsHTML	Format message as HTML.
formatAsText	Format message as text.
passesFilter	Determine whether message passes filter.

## Examples

### Create an Error Message

```
import mlreportgen.dom.*;
d = Document('test','html');

dispatcher = MessageDispatcher.getTheDispatcher;
l = addlistener(dispatcher,'Message',...
    @(src, evtdata) disp(evtdata.Message.formatAsText));

open(d);

dom.Document:2547 opening
dom.Document:2547 parsing template "B:/matlab/toolbox/shared/mlreportgen/dom/resources/templates,
dom.Document:2547 appended dom.TemplateText:2563
dom.Document:2547 appended dom.TemplateText:2566
dom.Document:2547 appended dom.TemplateText:2569
dom.Document:2547 moved to hole "#start#"

dispatch(dispatcher,ErrorMessage('invalid chapter',d));

dom.Document:2547 invalid chapter

p = Paragraph('Chapter ');
p.Tag = 'chapter title';
p.Style = {CounterInc('chapter'),...
    CounterReset('table'),WhiteSpace('pre')};
append(p,AutoNumber('chapter'));
append(d,p);

dom.Document:2547 appended chapter title

close(d);
```

```
dom.Document:2547 appended dom.TemplateText:2584  
dom.Document:2547 moved to hole "#end#"  
dom.Document:2547 closed
```

```
rptview(d.OutputPath);
```

Delete the listener to avoid duplicate reporting of message objects during a MATLAB® session.

```
delete(l);
```

## **See Also**

dispatch

## **Topics**

“Display Progress and Debugger Messages” on page 13-115

# mlreportgen.dom.ExternalLink class

**Package:** mlreportgen.dom

Hyperlink to a location outside of document

## Description

Defines a hyperlink to a location outside of the document.

## Construction

`externalLinkObj = ExternalLink(target, linkText)` creates a hyperlink to the specified target and having the specified link text. This constructor creates a text object (`mlreportgen.dom.Text`) to hold the link text.

`externalLinkObj = ExternalLink(target, linkText, linkTextStyleName)` creates a hyperlink with the specified link text and style name.

`externalLinkObj = ExternalLink(target, textObj)` creates a hyperlink to the specified target using the specified `Text` object.

## Input Arguments

### **target — Target of link**

character vector | `mlreportgen.dom.LinkTarget` object

The link target of the external link, specified as either a character vector (for a URL) or as an `mlreportgen.dom.LinkTarget` object.

### **linkText — Link text**

character vector

The text to use for the link text.

### **linkTextStyleName — Name of style for link text**

character vector

Name of style to use for the link text.

### **textObj — Text object containing link text**

`mlreportgen.dom.Text` object

Text object containing link text, specified by an `mlreportgen.dom.Text` object.

## Output Arguments

### **externalLinkObj — External link**

`mlreportgen.dom.ExternalLink` object

External link, represented by an `mlreportgen.dom.ExternalLink` object.

## Properties

### CustomAttributes — Custom attributes of document element

array of `mlreportgen.dom.CustomButton` objects

Custom attributes of this document element, specified as an array of `mlreportgen.dom.CustomButton` objects. The custom attributes must be supported by the output format.

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Style — Format specification

array of format objects

Format objects that specify the format of a document element.

### StyleName — Name of link style defined in the template

character vector

Name of link style defined in the template, specified as a character vector. The style specified by `styleName` must be defined in the template used to create the document to which the link is appended.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Target — Target URL of link

character vector

This read-only property displays the URL of the link target of this hyperlink.

## Methods

Method	Purpose
<code>append</code>	Append text or a <code>Text</code> , <code>Image</code> , or <code>CustomElement</code> object.
<code>clone</code> Use <code>ExternalLink.clone</code> in a similar way to how you use <code>Paragraph.clone</code> .	Copy the external link

## Examples

### Add an External Link

```
import mlreportgen.dom.*
d = Document('mydoc');

append(d,ExternalLink('https://www.mathworks.com/', 'MathWorks'));

close(d);
rptview(d.OutputPath);
```

### See Also

[mlreportgen.dom.LinkTarget](#) | [mlreportgen.dom.InternalLink](#)

### Topics

“Create Links” on page 13-82

## mlreportgen.dom.FirstLineIndent class

**Package:** mlreportgen.dom

Indent first line of paragraph

### Description

Indent first line of a paragraph.

### Construction

`firstLineIndentObj = FirstLineIndent()` creates an empty first line indentation format object.

`firstLineIndentObj = FirstLineIndent(width)` indents first line of paragraph by the specified amount.

`firstLineIndentObj = FirstLineIndent(style,width)` indents either the first line of the paragraph relative to the page margin or indents the subsequent lines relative to the page margin (hanging indentation).

### Input Arguments

#### **width — Width of indentation of first line of paragraph**

character vector

Width of indentation of the first line of the paragraph, specified in the form `valueUnits`. `Units` is an abbreviation for the units. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

#### **style — Type of indentation**

`'normal' | 'hanging'`

Type of indentation of the first line of the paragraph, specified as one of these values:

- `'normal'` (default) — indent relative to the page margin
- `'hanging'` — indent relative to the subsequent lines

### Output Arguments

#### **firstLineIndentObj — Indentation for first line of paragraph**

mlreportgen.dom.FirstLineIndent object

Indentation for first line of paragraph, represented by an `mlreportgen.dom.FirstLineIndent` object.

## Properties

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **Style — Type of indentation**

'normal' | 'hanging'

Type of indentation of the first line of the paragraph, specified as one of these values:

- 'normal' (default) — indent relative to the page margin
- 'hanging' — indent relative to the subsequent lines

### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### **Width — Amount of indentation**

character vector

Width of indentation of first line of paragraph in the form valueUnits. Units is an abbreviation for the units. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

## See Also

`mlreportgen.dom.Paragraph`

## Topics

“Report Formatting Approaches” on page 13-17

## mlreportgen.dom.FlowDirection class

**Package:** mlreportgen.dom

Direction of text or table column flow

### Description

Specifies the direction for text to flow across a page or the order of columns.

### Construction

`flowDirectionObj = FlowDirection()` causes text to flow from left to right and for the first column to be on the left side of a table.

`flowDirectionObj = FlowDirection(flow)` causes text to flow or column to appear in the specified direction (left-to-right or right-to-left).

### Input Arguments

#### **flow** — Direction for text to flow or table column ordering

'ltr' | 'rtl'

Direction for text to flow or for table columns to appear, specified as one of these values:

- 'ltr' — text flow or table column order is from left to right
- 'rtl' — text flow or table column order is from right to left

### Output Arguments

#### **flowDirectionObj** — Text flow direction or column order

mlreportgen.dom.FlowDirection object

Text flow or table column order, represented by an mlreportgen.dom.FlowDirection object.

### Properties

#### **Id** — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

#### **Tag** — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.



**Value — Text flow direction or column order direction**`'ltr' | 'rtl'`

Text flow direction or column order direction, specified as one of these values:

- 'ltr' — text flow or table column order is from left to right
- 'rtl' — text flow or table column order is from right to left

**Examples****Flow Text from Right to Left**

In this example, changing the text flow direction changes “stressed” into “desserts”.

```
import mlreportgen.dom.*;
doctype = 'docx';
d = Document('test',doctype);

p = Paragraph('desserts');
p.Style = {FlowDirection('rtl')};
append(d,p);

q = clone(p);
q.Style = {FlowDirection('ltr')};
append(d,q);

close(d);
rptview(d.OutputPath);
```

**See Also**

`mlreportgen.dom.Paragraph` | `mlreportgen.dom.TextOrientation`

**Topics**

“Create and Format Paragraphs” on page 14-67

## mlreportgen.dom.FontFamily class

**Package:** mlreportgen.dom

Font family

### Description

Properties of font family to be used to format document text.

### Construction

fontFamilyObj = FontFamily() creates a Times New Roman font family.

fontFamilyObj = FontFamily(fontStr) creates the specified font family.

### Input Arguments

#### **fontStr — Font family**

character vector

Font family, specified as a character vector.

### Output Arguments

#### **fontFamilyObj — Font family**

mlreportgen.dom.FontFamily object

Font family, represented by an mlreportgen.dom.FontFamily object.

### Properties

#### **BackupFamilyNames — Backup font families**

cell array

For HTML documents only. Cell array of character vectors specifying font families that a browser can use if the font family specified in FamilyName is not available on a system.

#### **ComplexScriptFamilyName — Font family for complex scripts**

character vector

For Word documents only. Font family to substitute in a locale that requires a complex script (such as Arabic) to render text, specified as a character vector.

#### **EastAsiaFamilyName — Font family for East Asian locales**

character vector

For Word documents only. Font family to substitute in an East Asian locale, such as China, Japan, or Korea, specified as a character vector.

#### **FamilyName — Font family to use**

character vector

Font family to use for document text, specified as a character vector.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**See Also**

mlreportgen.dom.FontSize | mlreportgen.dom.Text | mlreportgen.dom.Paragraph | mlreportgen.dom.CharEntity

**Topics**

“Report Formatting Approaches” on page 13-17

## mlreportgen.dom.FontSize class

**Package:** mlreportgen.dom

Font size

### Description

Specifies the size of a font.

### Construction

`fontSizeObj = FontSize()` creates a 12-point font.

`fontSizeObj = FontSize(sizeStr)` creates the specified font size.

### Input Arguments

#### **sizeStr** — Font size

'12pt' (default) | character vector

Font size, specified in the format `valueUnits`. `Units` is an abbreviation for the units. The following abbreviations are valid:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

### Output Arguments

#### **fontSizeObj** — Font size

mlreportgen.dom.FontSize object

Font size, represented by an `mlreportgen.dom.FontSize` object.

### Properties

#### **Id** — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

#### **Tag** — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Value — Font size**

'12pt' (default) | character vector

Font size, in the form `valueUnits`. Units is an abbreviation for the units. The following abbreviations are valid:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**See Also**

`mlreportgen.dom.FontFamily` | `mlreportgen.dom.Text` | `mlreportgen.dom.Paragraph`

**Topics**

“Report Formatting Approaches” on page 13-17

## mlreportgen.dom.EmbeddedObject class

**Package:** mlreportgen.dom mlreportgen.dom mlreportgen.dom

**Superclasses:** mlreportgen.dom.ExternalLink

Embed a file in a document

### Description

Use an object of the `mlreportgen.dom.EmbeddedObject` class to embed a file in a document. The advantage of embedding a file in a document, instead of linking to it, is that you can move the document without moving the target file.

You can append an `mlreportgen.dom.EmbeddedObject` object to objects of these DOM classes:

- `mlreportgen.dom.Document`
- `mlreportgen.dom.DocumentPart`
- `mlreportgen.dom.Paragraph`
- `mlreportgen.dom.TableEntry`
- `mlreportgen.dom.TableHeaderEntry`
- `mlreportgen.dom.ListItem`

The effect of appending an `EmbeddedObject` object to a DOM object depends on the document output type and the external file type, as described in this table.

Document Output Type	External File Type	Behavior
Word	xlsx, pptx, docx	Embeds the external file in the document and inserts an Object Linking and Embedding (OLE) link to the embedded file.  If you open the document without first opening it with <code>rptview</code> , the OLE link displays a placeholder image. After you open the document with <code>rptview</code> , the OLE link displays an image of the embedded file content. To see and edit the embedded file content, double-click the image.
Word	Any type except xlsx, pptx, or docx	Does not embed the file in the document. Inserts a hyperlink to the external file.

Document Output Type	External File Type	Behavior
PDF	Any	<p>Embeds the external file in the document and inserts an annotation that has a paper clip icon. To open the embedded file, double-click the icon.</p> <p>To see the paper clip icon, you must open the PDF outside of MATLAB. If you still do not see the paper clip icon, try opening the PDF using a different PDF viewer, such as Adobe Acrobat. If you need to open the PDF using a viewer that does not display the paper clip icon, instead of embedding the document, consider linking to the document by using an <code>mlreportgen.dom.External Link</code> object.</p>
HTML	Any	<p>Embeds the external file in the document and inserts a hyperlink to the embedded file.</p> <p>In an HTML document, the hyperlink works only if you open the document outside of MATLAB.</p>
Single-file HTML	Any	<p>Does not embed the file in the document. Inserts a hyperlink to the external file.</p> <p>The hyperlink works only if you open the document outside of MATLAB.</p>

The `mlreportgen.dom.EmbeddedObject` class is a handle class.

#### Class Attributes

<code>ConstructOnLoad</code>	<code>true</code>
<code>HandleCompatible</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`embObj = mlreportgen.dom.EmbeddedObject()` creates an empty `EmbeddedObject` object. To specify the file to embed, use the `Target` property.

`embObj = mlreportgen.dom.EmbeddedObject(path)` sets the `Target` property to the path of the file to embed.

`embObj = mlreportgen.dom.EmbeddedObject(path,linkText)` also specifies the link text. The link text applies only when the reference to a file is a hyperlink. This constructor creates an `mlreportgen.dom.Text` object to hold the link text.

`embObj = mlreportgen.dom.EmbeddedObject(path,linkText,styleName)` specifies the name of the template-defined style to use for the link text. This constructor creates an `mlreportgen.dom.Text` object to hold the link text and sets the `StyleName` property of the `Text` object to `styleName`.

`embObj = mlreportgen.dom.EmbeddedObject(path,linkTextObj)` uses an `mlreportgen.dom.Text` object to specify the link text.

### Input Arguments

#### **path — Path of file to embed**

character vector | string scalar

Path of the file to embed, specified as a character vector or string scalar.

#### **linkText — Link text**

character vector | string scalar

Link text, specified as a character vector or string scalar. The link text applies only when the reference to a file is a hyperlink.

#### **linkTextObj — Link text**

`mlreportgen.dom.Text` object

Link text, specified as an `mlreportgen.dom.Text` object. The link text applies only when the reference to a file is a hyperlink.

#### **styleName — Name of style for link text**

character vector | string scalar

Name of the template-defined style to use for the link text, specified as a character vector or string scalar.

## Properties

#### **Target — Path of file to embed in document**

character vector | string scalar

Path of the file to embed in the document, specified as a character vector or string scalar.



**StyleName — Name of style to use for link**

character vector | string scalar

Name of the style to use for the link, specified as a character vector or string scalar. The style specified by the `StyleName` property must be defined in the template that is used to create the document to which the `EmbeddedObject` is appended. The `StyleName` property applies only when the reference to a file is a hyperlink.

**Style — Formats that define link style**

cell array of DOM format objects

Formats that define the style of a link, specified as a cell array of DOM format objects. Formats specified in the `Style` property override formats specified by a template-defined style. The `Style` property applies only when the reference to a file is a hyperlink.

**CustomAttributes — Custom attributes of link**

[] | cell array of mlreportgen.dom.CustomAttribute objects

Custom attributes of a link, specified as a cell array of `mlreportgen.dom.CustomAttribute` objects. The output format must support the custom attributes. The `CustomAttributes` property applies only when the reference to a file is a hyperlink.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

## Methods

### Public Methods

append	Append text, an image, or a custom element to the reference to the file in a document. Use <code>append</code> with an <code>mlreportgen.dom.EmbeddedObject</code> object the same way as you use <code>append</code> with an <code>mlreportgen.dom.ExternalLink</code> object.  The <code>append</code> method applies only when the reference to a file is a hyperlink.
clone	<code>clone = clone(embeddedObj)</code> creates a copy of this <code>mlreportgen.dom.EmbeddedObject</code> object.  To repeat the content in a document, use <code>clone</code> . Do not append the same object to a document more than one time.

## Examples

### Embed a Word Document in a Different Word Document

Create a Word document, `CompanyInfo.docx`, and embed it in a second Word document, `CompanyReport.docx`.

```
import mlreportgen.dom.*

doc1 = Document('CompanyInfo','docx');
append(doc1, 'XYZ, Inc., makes widgets. ');
close(doc1);

doc1Path = doc1.OutputPath;

doc2 = Document('CompanyReport','docx');
open(doc2);

para = append(doc2, Paragraph('About XYZ, Inc. '));
append(doc2, EmbeddedObject(doc1Path));
close(doc2);
rptview(doc2);
```

The document, `CompanyReport.docx`, displays an image of the embedded file.

About XYZ, Inc.



To open the embedded document, double-click the image.

### Embed a PDF in an HTML Document

Create a PDF, `bio.pdf`, and embed it in an HTML document, `author.html`. Specify the link text in the `mlreportgen.dom.EmbeddedObject` constructor. Make the link green.

```
import mlreportgen.dom.*
doc1 = Document('bio','pdf');
para = 'The author was born in New York, NY in 1970.';
append(doc1,para);
close(doc1);

doc2 = Document('author','html');
doc2.PackageType = 'unzipped';

embedObj = EmbeddedObject('bio.pdf','Author Biography');
embedObj.Style={Color('green')};
p = Paragraph();
append(p,embedObj);
append(doc2,p);

close(doc2);
rptview(doc2);
```

The document `author` displays a link to the embedded document.

### Author Biography

To open the embedded document, open `author/root.html` outside of MATLAB.

## Alternative Functionality

### `mlreportgen.dom.DOCXSubDoc`

To insert Word documents into another document, you can use objects of the `DOCXSubDoc` or `EmbeddedObject` class. To decide which class to use, consider these guidelines:


- Use `DOCXSubDoc` objects if you want the parent document to display links to subdocuments initially. You might want to display links initially when the parent document links to a set of subdocuments in an outline view. Use `EmbeddedObject` objects if you want the parent document to display the subdocument contents regardless of whether you open the parent document in Word or by using `rptview`.
- Use `EmbeddedObject` objects if you want to move the parent document without moving the subdocuments.

## Compatibility Considerations

### Appearance of link to embedded file in a PDF report

*Behavior changed in R2020b*

Starting in R2020b, if you use `mlreportgen.dom.EmbeddedObject` to embed a file in a PDF report, the link to the embedded file is inserted into the PDF as an annotation that has a paper clip icon. For example:

About XYZ, Inc.  
  
[CompanyInfo.pdf](#)

To open the embedded file, double-click the paper clip icon. In previous releases, a hyperlink to the embedded file was inserted into the PDF. For example:

About XYZ, Inc.  
[CompanyInfo.pdf](#)

### See Also

`mlreportgen.dom.ExternalLink` | `mlreportgen.dom.Text`

### Topics

“Report Formatting Approaches” on page 13-17

**Introduced in R2020a**

# mlreportgen.dom.FOProperties class

**Package:** mlreportgen.dom

Array of FO properties for formatting PDF output

## Description

Creates an object that contains one or more Extensible Stylesheet Language (XSL) formatting objects (FO) for PDF output. Specify the formats using `mlreportgen.dom.FOProperty`. For information on FO properties, see [www.w3.org/2002/08/XSLFOsummary.html](http://www.w3.org/2002/08/XSLFOsummary.html).

---

**Note** Use `FOProperties` objects only for FO properties not supported by other DOM format objects. Most DOM format objects work for all output types. Using `FOProperty` objects makes your report application specific to PDF output.

---

## Construction

`props = FOProperties(prop)` creates an `FOProperties` object based on `mlreportgen.dom.CSSProperty` objects. The `mlreportgen.dom.FOProperty` object specifies the CSS format to use and its value.

## Input Arguments

### prop — FO property

`mlreportgen.dom.FOProperty` object | array of `mlreportgen.dom.FOProperty` objects | cell array of `mlreportgen.dom.FOProperty` objects

FO property, specified as an `mlreportgen.dom.FOProperty` object or as an array or cell array of `mlreportgen.dom.FOProperty` objects.

## Output Arguments

### props — FO properties

`mlreportgen.dom.FOProperties` object

FO properties, returned as an `mlreportgen.dom.FOProperties` object.

## Properties

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Parent — Parent of document element

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

### Properties — FO properties

array of `mlreportgen.dom.FOProperty` objects

FO properties, specified as an array of `mlreportgen.dom.FOProperty` objects.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Examples

### Apply FO Property to a List

This example shows how to apply an FO property to a `List` object. Using the DOM API, you can set a page break property on a paragraph using `PageBreakBefore`. However, you cannot use the `PageBreakBefore` property on a list. Instead, for PDF output, you can use the FO property `'break-before'` with the value `'page'`.

```
import mlreportgen.dom.*

d = Document('Break Before List', 'pdf');

p = Paragraph('First Page');
p.Style = {PageBreakBefore};
append(d, p);

p = Paragraph('Second Page');
p.Style = {PageBreakBefore};
append(d, p);

list = UnorderedList({'Earl Grey', 'Jasmine', 'Honeybush'});
list.Style = {FOProperties(FOProperty('break-before', 'page'))};
append(d, list);

close(d);
rptview(d.OutputPath);
```

### See Also

`mlreportgen.dom.FOProperty` | `mlreportgen.dom.Paragraph` | `mlreportgen.dom.Page`

### Topics

“Report Formatting Approaches” on page 13-17

### Introduced in R2016a

# mlreportgen.dom.FOProperty class

**Package:** mlreportgen.dom

FO property for PDF output

## Description

Creates an object that specifies an XML Style Sheet Language (XSL) Formatting Object (FO) property. The DOM API uses FO objects to format PDF output. Use this object with `mlreportgen.FOProperties` to apply FO properties not supported by DOM format objects. For more information, see [w3.org/2002/08/XSLFOsummary.html](http://w3.org/2002/08/XSLFOsummary.html).

## Construction

`prop = FOProperty(Name, Value)` creates an FO format property having the specified name and value.

## Input Arguments

### Name — FO property name

character vector

FO property name, specified as a character vector.

### Value — Property value

character vector

Property value for the corresponding property name, specified as a character vector.

## Output Arguments

### prop — FO format object

mlreportgen.dom.FOproperty object

FO format object, returned as an `mlreportgen.dom.FOProperty` object.

## Properties

### Children — Children of document element

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Name — FO property name**

character vector

FO property name, specified as a character vector.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Value — Property value**

character vector

Property value for the corresponding property name, specified as a character vector.

## Examples

**Use FO Property to Break a Page on List**

This example shows how to apply an FO property to a List object. Using the DOM API, you can set a page break property on a paragraph using PageBreakBefore. However, you cannot use the PageBreakBefore property on a list. Instead, for PDF output, you can use the FO property 'break-before' with the value 'page'.

```
import mlreportgen.dom.*

d = Document('Break Before List', 'pdf');

listbreak = FOProperty('break-before', 'page');
p = Paragraph('First Page');
p.Style = {PageBreakBefore};
append(d,p);

p = Paragraph('Second Page');
p.Style = {PageBreakBefore};
append(d,p);

list = UnorderedList({'Earl Grey', 'Jasmine', 'Honeybush'});
list.Style = {FOProperties(listbreak)};
append(d,list);
```



```
close(d);  
rptview(d.OutputPath);
```

### **See Also**

[mlreportgen.dom.FOProperties](#) | [mlreportgen.dom.CSSProperties](#) |  
[mlreportgen.dom.PageBreakBefore](#)

### **Topics**

“Report Formatting Approaches” on page 13-17

### **Introduced in R2016a**

## mlreportgen.dom.FormalTable class

**Package:** mlreportgen.dom

Formal table

### Description

Defines a formal table, which is a table that has a body and optionally a table header, a table footer, or both. The table header, body, and footer are mlreportgen.dom.TableHeader, mlreportgen.dom.TableBody, and mlreportgen.dom.TableFooter objects, respectively.

The mlreportgen.dom.FormalTable class is a handle class.

### Creation

#### Description

formalTableObj = FormalTable() creates an empty formal table. Use this constructor as the starting point to create a formal table from scratch.

formalTableObj = FormalTable(ncols) creates an empty formal table having the specified number of columns.

formalTableObj = FormalTable(body) creates a formal table with the body content specified. The constructor converts basic MATLAB types to corresponding DOM objects. For example, the constructor converts character vectors to mlreportgen.dom.Text objects.

formalTableObj = FormalTable(body, styleName) creates a formal table having the specified body content and style.

formalTableObj = FormalTable(header, body) creates a formal table with a header and a body using the specified contents, and an empty footer.

formalTableObj = FormalTable(header, body, styleName) creates a formal table using the specified content and style. The table has an empty footer.

formalTableObj = FormalTable(header, body, footer) creates a formal table with the specified content for the body, header, and footer.

#### Input Arguments

##### **ncols** — Number of columns in table

numeric value

Number of columns in a table, specified as a numeric value.

Data Types: double

##### **body** — Table body content

two-dimensional numeric array | two-dimensional categorical array | two-dimensional cell array

Table body content, specified as:

- A two-dimensional numeric array
- A two-dimensional categorical array
- A two-dimensional cell array that can contain:
  - Character vectors
  - One- or two-dimensional cell array
  - `double`
  - `mlreportgen.dom.Text` object
  - `mlreportgen.dom.Paragraph` object
  - `mlreportgen.dom.Image` object
  - `mlreportgen.dom.Table` object
  - `mlreportgen.dom.FormalTable` object
  - `mlreportgen.dom.OrderedList` object
  - `mlreportgen.dom.UnorderedList` object
  - `mlreportgen.dom.ExternalLink` object
  - `mlreportgen.dom.InternalLink` object
  - `mlreportgen.dom.CharEntity` object

#### **styleName — Style for table**

character vector

The style specified by `styleName` must be defined in the template used to create the document that contains this table.

#### **header — Header content**

two-dimensional numeric array | two-dimensional cell array of character vectors

The cell array may contain:

- Character vectors
- One- or two-dimensional cell array
- `double`
- `mlreportgen.dom.Text` object
- `mlreportgen.dom.Paragraph` object
- `mlreportgen.dom.Image` object
- `mlreportgen.dom.Table` object
- `mlreportgen.dom.FormalTable` object
- `mlreportgen.dom.OrderedList` object
- `mlreportgen.dom.UnorderedList` object
- `mlreportgen.dom.ExternalLink` object
- `mlreportgen.dom.InternalLink` object
- `mlreportgen.dom.CharEntity` object

**footer — Footer content**

two-dimensional numeric array | two-dimensional cell array of character vectors

The cell array may contain:

- Character vector
- One- or two-dimensional cell array
- double
- `mlreportgen.dom.Text` object
- `mlreportgen.dom.Paragraph` object
- `mlreportgen.dom.Image` object
- `mlreportgen.dom.Table` object
- `mlreportgen.dom.FormalTable` object
- `mlreportgen.dom.OrderedList` object
- `mlreportgen.dom.UnorderedList` object
- `mlreportgen.dom.ExternalLink` object
- `mlreportgen.dom.InternalLink` object
- `mlreportgen.dom.CharEntity` object

**Properties****BackgroundColor — Background color**

[] (default) | character vector | string scalar

Background color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `BackgroundColor` property adds a corresponding `mlreportgen.dom.BackgroundColor` format object to the `Style` property. Setting the `BackgroundColor` property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

**Body — Table body**`mlreportgen.dom.TableBody` object

The table constructor creates a table body object and assigns it to this property when the formal table is constructed. You cannot subsequently set this property. However, you can append content to the table body and set its properties via this property.

**Border — Type of border to draw**

'solid' | 'single' | 'dashed' | 'none' | ...

Type of border to draw, specified as one of the values in the table.

<b>Border Value</b>	<b>Description</b>	<b>Supported Output Types</b>
'dashed'	Dashed line	All output types
'dashdotstroked'	Line with alternating diagonal dashes and dot	Word
'dashsmallgap'	Dashed line with a small gap between dashes	Word
'dotted'	Dotted line	All output types
'dotdash'	Line with alternating dots and dashes	Word
'dotdotdash'	Line with alternating double dots and a dash	Word
'double'	Double line	All output types
'doublewave'	Double wavy line	Word
'groove'	3-D effect grooved line	HTML and PDF
'hidden'	No line  The 'none' border type also produces no line. However, conflicting borders are handled differently for 'hidden' types than for 'none' types. The 'hidden' border type takes precedence over a conflicting border type. A conflicting border type takes precedence over the 'none' type.	HTML and PDF
'inset'	3-D effect line	All output types
'none'	No line  The 'hidden' border type also produces no line. However, conflicting borders are handled differently for 'hidden' types than for 'none' types. The 'hidden' border type takes precedence over a conflicting border type. A conflicting border type takes precedence over the 'none' type.	All output types
'outset'	3-D effect line	All output types
'ridge'	3-D effect ridged line	HTML and PDF
'single'	Single line	Word
'solid'	Single line	HTML and PDF
'thick'	Thick line	Word

Border Value	Description	Supported Output Types
'thickthinlargegap'	Dashed line with alternating thick and thin dashes with a large gap	Word
'thickthinmediumgap'	Dashed line with alternating thick and thin dashes with a medium gap	Word
'thickthinsmallgap'	Dashed line with alternating thick and thin dashes with a small gap	Word
'thinthicklargegap'	Dashed line with alternating thin and thick dashes with a medium gap	Word
'thinthickmediumgap'	Dashed line with alternating thin and thick dashes, with a medium gap	Word
'thinthicksmallgap'	Dashed line with alternating thin and thick dashes with a small gap	Word
'thinthickthinlargegap'	Dashed line with alternating thin and thick dashes with a large gap	Word
'thinthickthinmediumgap'	Dashed line with alternating thin and thick dashes with a medium gap	Word
'thinthickthinsmallgap'	Dashed line with alternating thin and thick dashes with a small gap	Word
'threedemboss'	Embossed effect line	Word
'threedengrave'	Engraved effect line	Word
'triple'	Triple line	Word
'wave'	Wavy line	Word

### BorderCollapse — Collapse borders of adjacent cells into single border (HTML only)

'on' | 'off'

A value of 'on' collapses borders of adjacent cells into a single border. A value of 'off' keeps borders of adjacent cells.

### BorderColor — Border color

character vector

Border color, specified as either:

- Name of a color. The name must be a CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.

- A hexadecimal RGB (truecolor) value as #RRGGBB. For example, #0000ff is a shade of blue.

### **BorderWidth — Table border width**

character vector

Table border width, specified in the form `valueUnits`, where `Units` is an abbreviation for the units. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

### **Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

### **ColSep — Style of line separating columns**

character vector

The style of the line separating the columns of a table or table section (header, body, footer), as specified by an `mlreportgen.dom.ColSep` object.

See the description of the `Border` property for a description of the possible values.

### **ColSepColor — Color of line separating columns**

character vector

Color of line separating columns, specified as either:

- Name of a color. The name must be a CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- A hexadecimal RGB (truecolor) value as #RRGGBB. For example, #0000ff is a shade of blue.

### **ColSepWidth — Width of line separating table columns**

character vector

Width of the line separating table columns, in the form `valueUnits`. Use one of these abbreviations for the `Units`:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

For example, for a column separator of 3 points, set the `ColSepWidth` property to `'3pt'`.

**ColSpecGroups — Properties of group of columns in table**

array of `mlreportgen.dom.TableColSpecGroup` objects

An array of `mlreportgen.dom.TableColSpecGroup` objects that specifies the width, alignment, and other properties of a group of columns. The first object applies to the first group of columns, the second object to the second group, and so on. Specify the number of columns belonging to each group using the `Span` property of the `TableColSpecGroup` object. For example, if the first object has a span of 2, it applies to the first two columns. If the second group has a span of 3, it applies to the next three columns, and so on.

**CustomAttributes — Custom attributes for document element**

array of `mlreportgen.doc.CustomAttribute` objects

The custom attributes must be supported by the output type of the document to which this document element is appended.

**FlowDirection — Column flow direction**

'ltr' | 'rtl'

Column flow direction, specified as:

- 'ltr' — Flow from left to right (column 1 is to the left in the table).
- 'rtl' — Flow from right to left (column 1 is to the right in the table).

**Footer — Footer for this table**

`mlreportgen.dom.TableFooter` object

The table constructor creates a table footer object and assigns it to this property when the formal table is constructed. You cannot subsequently set this property. However, you can append content to the table body and set its properties via this property.

**HAlign — Horizontal alignment of this table**

'center' | 'left' | 'right'

Horizontal alignment of this table, specified as one of these values:

- 'center'
- 'left'
- 'right'

---

**Note** To prevent the overflow of large tables in PDF output, set the `Width` property.

---

**Header — Table header**

`mlreportgen.dom.TableHeader` object

The table constructor creates a table header object and assigns it to this property when the formal table is constructed. You cannot subsequently set this property. However, you can append content to the table body and set its properties via this property.

**Id — ID for this document element**

character vector | string scalar



ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **OuterLeftMargin — Left margin (indentation) of document element**

character vector

Left indentation in the form `valueUnits`. `Units` is an abbreviation for the units. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

### **Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

### **RowSep — Style of lines separating rows**

character vector

The style of a line separating the rows of a table or table section (header, body, or footer).

See the description of the `Border` property for a description of the possible values.

### **RowSepColor — Color of row separator**

character vector

You can specify:

- Name of a color. The name must be a CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

### **RowSepWidth — Width of row separator**

character vector

Width of the row separator, specified in the form `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

### **Style — Format for table**

array of format objects

Array of format objects (such as `Bold` objects) that specify the format for this table.

This property overrides corresponding formats defined by the style sheet style specified by the `StyleName` property.

### **StyleName — Style in document or document part style sheet**

character vector

Name of a style specified in the style sheet of the document or document part to which this table is appended

The style that specifies the appearance of this table in the output document, for formats not specified by `Style` property.

You can set the `StyleName` property of any formal table section. Setting `StyleName` overrides the style specified by the formal table itself. However, if you do this for a Word document, you must explicitly specify the width of each column in a section to ensure that all sections have the same width. Word, unlike HTML and PDF, has no built-in support for formal tables. To handle this, the DOM interface represents a formal table as three tables, one for each section, embedded in a 3x1 table.

### **TableEntriesStyle — Style to use for table entries**

cell array

Cell array of format objects that specify the format for table entries.

### **TableEntriesInnerMargin — Inner margin for table entries**

character vector

The inner margin is the margin between table cell content and the cell borders in the form `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### **Width — Table width**

character vector

A percentage (for example, `'100%'`) of the page width (minus margins for Word reports) or a number of units of measurement, having the format `valueUnits`. `Units` is an abbreviation for the units. These are valid abbreviations:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

## Methods

### Public Methods

Method	Purpose
append  Use <code>FormalTable.append</code> similar to how you use <code>TableRow.append</code> .	Append a row of table entries to table
appendFooterRow	Append row to table footer
appendHeaderRow	Append row to table header
clone  Use <code>FormalTable.clone</code> the same way you use <code>Paragraph.clone</code> .	Copy the table

### See Also

[mlreportgen.dom.TableRow](#) | [mlreportgen.dom.TableColSpec](#) |  
[mlreportgen.dom.ResizeToFitContents](#) | [mlreportgen.dom.TableEntry](#) |  
[mlreportgen.dom.Table](#) | [mlreportgen.dom.TableBody](#) | [mlreportgen.dom.TableFooter](#) |  
[mlreportgen.dom.TableHeader](#)

### Topics

“Create Informal Tables” on page 13-64

### Introduced in R2014b

## mlreportgen.dom.Group class

**Package:** mlreportgen.dom

Group of document objects

### Description

Group of document objects that you can append multiple times in a document without you having to clone the group. When you append a group to a document, the DOM interface clones the group.

---

**Tip** You can use `mlreportgen.dom.Group` and `mlreportgen.dom.Container` objects to produce collections of document elements.

- Use a group object to append the same content in multiple places in a document without having to clone the group. Group objects do not have a `Style` property for using the same applicable styles to all document elements in the group.
  - Use a container object to create a `div`, `section`, or `article` container element and to use the same applicable styles to all document elements in the container. To append the same container object contents in multiple places in a document, use the `mlreportgen.dom.Container.clone` method.
- 

### Construction

`groupObj = Group()` creates an empty group.

#### Output Arguments

**groupObj** — Group of document objects

mlreportgen.dom.Group object

Group of document objects, represented by an `mlreportgen.dom.Group` object.

### Properties

**Id** — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Children** — Children of document element

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**Parent** — Parent of document element

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Methods**

Method	Purpose
append	Append a DOM object to the group

**See Also**

mlreportgen.dom.Container

**Topics**

“Add Content as a Group” on page 13-12

## mlreportgen.dom.HAlign class

**Package:** mlreportgen.dom

Specify horizontal alignment of document object

### Description

Specifies horizontal alignment of a document object.

The mlreportgen.dom.HAlign class is a handle class.

### Creation

#### Description

alignObj = HAlign() creates an alignment object having the value 'left'.

alignObj = HAlign(value) creates an alignment object having the specified value.

#### Input Arguments

##### value — Horizontal alignment

'center' | 'left' | 'right' | 'justify'

Horizontal alignment of a document object, specified as one of these values:

- 'center'
- 'left'
- 'right'
- 'justify'

### Properties

#### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

#### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Value — Horizontal alignment**`'center' | 'left' | 'right' | 'justify'`

Horizontal alignment, specified as one of these values:

- `'center'` — Center object between the sides of the container
- `'left'` — Align object to the left side of the container
- `'right'` — Align object to the right side of the container
- `'justify'` — Align text to the left and right sides of the container, adjusting word and letter spacing and hyphenating as necessary (if hyphenation is enabled).

**See Also**`mlreportgen.dom.VAlign`**Topics**

“Report Formatting Approaches” on page 13-17

**Introduced in R2014b**

## mlreportgen.dom.XRef class

**Package:** mlreportgen.dom

Cross-reference element for DOCX or PDF report

### Description

Use an object of the mlreportgen.dom.XRef class to cross-reference a target in a DOCX or PDF report.

The mlreportgen.dom.XRef class is a handle class.

### Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

xRefObj = mlreportgen.dom.XRef() creates an mlreportgen.dom.XRef object with an empty Target property. Set the Target property before you add the XRef object to a report.

xRefObj = mlreportgen.dom.XRef(target) creates an XRef object with the link ID, target.

## Properties

### Target — Link ID of target object

[] (default) | character array | string scalar

Link ID of the target object to cross-reference, specified as a character array or string scalar. Set the Target property to the link ID of the mlreportgen.dom.LinkTarget object you want to cross-reference.

---

**Note** Use the mlreportgen.utils.normalizeLinkID function to generate a valid link target ID that conforms to the limitations of PDF and Microsoft Word documents.

---

### Attributes:

GetAccess public  
SetAccess public

Data Types: char | string

### Style — Format objects

cell array



Format objects, specified as a cell array of `mlreportgen.dom` format objects. Add one or more format objects to customize the appearance of the cross-reference.

**Attributes:**

GetAccess	public
SetAccess	public

**StyleName — Name of style to apply from style sheet**

[] (default) | character array | string scalar

Name of the style to use to render the cross-reference, specified as a string scalar or character vector. The style must be defined in the style sheet of the template associated with the document that contains the cross-reference. To learn more about using style sheets, see “Use Style Sheet Styles” on page 13-19.

**Attributes:**

GetAccess	public
SetAccess	public

Data Types: char | string

**Parent — Object to which cross-reference object is appended**

[] (default) | `mlreportgen.dom.Paragraph` object

Object to which the cross-reference object is appended, specified as an `mlreportgen.dom.Paragraph` object.

---

**Note** You can append an XRef object only to an object of the class `mlreportgen.dom.Paragraph`.

---

**Attributes:**

GetAccess	public
SetAccess	public

Data Types: `mlreportgen.dom.Paragraph`

**Children — Objects appended to cross-reference object**

[] (default) | `mlreportgen.dom.CustomElement` array

Objects appended to the XRef object, specified as an `mlreportgen.dom.CustomElement` array.

---

**Note** You can append only objects of the class `mlreportgen.dom.CustomElement` to an XRef object.

---

**Attributes:**

GetAccess	public
SetAccess	public

Data Types: `mlreportgen.dom.CustomElement`

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**CustomAttributes — Custom attributes of document element**array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this document element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. The custom attributes must be supported by the output format.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Methods****Public Methods**

<code>clone</code>	<code>xRefCopy = clone(xRefObj)</code> creates a copy of <code>xRefObj</code> .
--------------------	---

**Examples****Use Cross-Reference Elements in a PDF Report**

This example shows how to use cross-reference elements in a generated PDF report.

Import these packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
import mlreportgen.report.*
```

Create an `mlreportgen.report.Report` object of type PDF.

```
rpt = Report("Cross-ref_in_PDF_example", "pdf");
```

Create an `mlreportgen.report.Chapter` object for the introduction chapter, and an `mlreportgen.dom.Paragraph` object for the content of the introduction chapter.

```
chapter1 = Chapter("Introduction");
para = Paragraph();
para.WhiteSpace = "preserve";
```

Use the function `normalizeLinkID` to generate a valid unique link ID, then use this link ID to create an `mlreportgen.dom.LinkTarget` object.

```
linkID = mlreportgen.utils.normalizeLinkID("myLinkID");
linkTarget = LinkTarget(linkID);
```

Create two reference objects, an `mlreportgen.dom.XRef` and an `mlreportgen.dom.PageRef`, with the same link ID you used to create the `LinkTarget` object. You must use the same link ID for all three objects to link the reference objects to the link target object.

```
xRefObj = XRef(linkID);
pageRefObj = PageRef(linkID);
```

Customize the appearance of the cross-reference object by adding an `mlreportgen.dom.Italic` object to the `Style` property.

```
xRefObj.Style{end+1} = Italic;
```

Set the `IsXRefTarget` property of the `LinkTarget` object to `true`. This is necessary only in PDF reports.

```
linkTarget.IsXRefTarget = true;
```

Fill the paragraph with content that includes the cross-reference and the page reference objects.

```
append(para, "For more information see ");
append(para, xRefObj);
append(para, " on page ");
append(para, pageRefObj);
append(para, " for more information.");
```

Append the paragraph object to the chapter object, then append the chapter object to the report.

```
append(chapter1, para)
append(rpt, chapter1);
```

Create another `Chapter` object for the information chapter.

```
chapter2 = Chapter();
```

Append the text "Information" to the link target object and set the title of the chapter object to the link target object.

```
linkTarget.append("Information");
chapter2.Title = linkTarget;
```

Append the second chapter object to the report. Then close and view the report.

```
append(rpt, chapter2);
close(rpt);
rptview(rpt);
```

When the report opens, test the cross-reference and page reference elements.

## See Also

`mlreportgen.dom.LinkTarget` | `mlreportgen.dom.ExternalLink` |  
`mlreportgen.dom.InternalLink` | `mlreportgen.dom.PageRef` |  
`mlreportgen.report.TableOfContents` | `mlreportgen.utils.normalizeLinkID`

**Topics**

“Use Style Sheet Styles” on page 13-19

**Introduced in R2022a**

# mlreportgen.dom.Heading class

**Package:** mlreportgen.dom

Heading paragraph using variable level

## Description

Create a heading paragraph at a particular level or whose level is calculated during report generation.

---

**Tip** Use this class if you need to determine the heading level at runtime. Otherwise, you can use the `Heading1`, `Heading2`, etc. classes to avoid having to set the level explicitly.

---

## Construction

`headingObj = Heading(level)` creates an empty heading at the specified level.

`headingObj = Heading(level, text)` creates the specified level heading containing the specified text.

`headingObj = Heading(level, text, styleName)` creates the specified level heading containing the specified text and using the specified style.

`headingObj = Heading(level, domObj)` creates the specified level heading containing the specified DOM object.

## Input Arguments

### level — Heading level

numeric value | variable

Heading level, specified as a numeric value or variable. Use a variable to use a value calculated during report generation to determine the level number.

Data Types: double

### text — Heading text

character vector

Text to use for the heading, specified as a character vector.

### styleName — Style for text

character vector

The style specified by `styleName` must specify a paragraph style defined in the template used to create the document to which this heading is appended.

### domObj — DOM object to include in heading

mlreportgen.dom.ExternalLink object | mlreportgen.dom.Image object |  
mlreportgen.dom.InternalLink object | mlreportgen.dom.LinkTarget object |  
mlreportgen.dom.Text object

DOM object to include in the heading, specified as any of these DOM object types:

- ExternalLink
- Image
- InternalLink
- LinkTarget
- Text

### Output Arguments

#### **headingObj** — Heading paragraph

`mlreportgen.dom.Heading` object

Heading paragraph, represented by an `mlreportgen.dom.Heading` object.

## Properties

#### **BackgroundColor** — Background color

[] (default) | character vector | string scalar

Background color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form `#RRGGBB`.

Setting the `BackgroundColor` property adds a corresponding `mlreportgen.dom.BackgroundColor` format object to the `Style` property. Setting the `BackgroundColor` property to an empty value removes the object.

Example: `'blue'`

Example: `'#0000ff'`

#### **Bold** — Whether to use bold for text

[] (default) | true or 1 | false or 0

Whether to use bold for text, specified as a numeric or logical 1 (true) or 0 (false). To make text bold, set `Bold` to true or 1.

Setting the `Bold` property adds a corresponding `mlreportgen.dom.Bold` format object to the `Style` property. Setting the `Bold` property to an empty value removes the object.

#### **Color** — Text color

[] (default) | character vector | string scalar

Text color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form `#RRGGBB`.

Setting the `Color` property adds a corresponding `mlreportgen.dom.Color` format object to the `Style` property. Setting the `Color` property to an empty value removes the object.

Example: `'blue'`

Example: `'#0000ff'`

**CustomAttributes — Custom attributes of document element**array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this document element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. The custom attributes must be supported by the output format.

**FirstLineIndent — Indentation amount for first line of paragraph**

[] | character vector | string scalar

Indentation amount for the first line of this paragraph, specified as a character vector or string scalar that consists of number followed by an abbreviation for a unit of measurement. For example, '0.5in' specifies one-half inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

The indentation amount is the amount by which to indent the first line of this paragraph relative to the succeeding lines. To create a hanging indent where all of the lines are indented except for the first line, use a negative number.

Setting the `FirstLineIndent` property adds a corresponding `mlreportgen.dom.FirstLineIndent` format object to the `Style` property for this document element. Setting the `FirstLineIndent` property to an empty value removes the object.

Example: '0.5in'

**FontFamilyName — Name of font family for text**

[] | character vector | string scalar

Name of a font family to use for text, specified as a character vector or string scalar.

Setting the `FontFamilyName` property adds a corresponding `mlreportgen.dom.FontFamily` format object to the `Style` property. Setting the `FontFamilyName` property is the same as setting the `FontName` property of an `mlreportgen.dom.FontFamily` object. Setting the `FontFamilyName` property to an empty value removes the object.

To specify substitutions for the font, do not specify the font family by setting this property. Instead create and add an `mlreportgen.dom.FontFamily` object to the `Style` property.

Example: 'Courier New'

**FontSize — Font size for text**

character vector

Font size for text, in the form `valueUnits`, where `Units` is an abbreviation for the units. Use one of these abbreviations for the units.

- px — pixels (default)

- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points
- `px` — pixels

### **HALign** — Horizontal alignment of paragraph

`'center'` | `'distribute'` | ...

Horizontal alignment for this paragraph, relative to page margins or table cell borders, specified as one of the values in this table.

Value	Description	Supported Output Types
<code>'center'</code>	Center the paragraph	All
<code>'distribute'</code>	Distribute all characters equally	Word
<code>'justify'</code>	Align left side of paragraph on left side of page or table entry, and right side of paragraph on the right side of the page or table entry.	All
<code>'KashidaHigh'</code>	Use widest Kashida length.  Kashida is a type of justification used for some cursive scripts, such as Arabic and Persian.	Word
<code>'KashidaLow'</code>	Use lowest Kashida length	Word
<code>'KashidaMedium'</code>	Use medium Kashida length	Word
<code>'left'</code>	Align paragraph left	All
<code>'right'</code>	Align paragraph right	All
<code>'ThaiDistribute'</code>	Thai language justification	Word

Setting the `HALign` property adds a corresponding `mlreportgen.dom.HAlign` format object to the `Style` property for this document element. Setting the `HALign` property to an empty value removes the object.

### **Id** — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **Italic** — Whether to use italic for text

`[]` (default) | `true` or `1` | `false` or `0`

Whether to use italic for text, specified as a numeric or logical `1` (`true`) or `0` (`false`). To make text italic, set `Italic` to `true` or `1`.



Setting the `Italic` property adds a corresponding `mlreportgen.dom.Italic` format object to the `Style` property. Setting the `Italic` property to an empty value removes the object.

### **OuterLeftMargin — Left margin (indentation) of document element**

character vector

Left indentation in the form `valueUnits`. `Units` is an abbreviation for the units. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

### **OutlineLevel — Outline level of paragraph**

`[]` (default) | integer

Outline level of the paragraph, specified as an integer. Setting the `OutlineLevel` property causes this paragraph to be included in automatically generated outlines, such as a table of contents. The value specifies the level of the paragraph in the outline. For example, to make a paragraph appear at the top level in an outline, set the `OutlineLevel` property to 1.

Setting the `OutlineLevel` property adds a corresponding `mlreportgen.dom.OutlineLevel` format object to the `Style` property. Setting the `OutlineLevel` property to an empty value removes the object.

### **Strike — Text strikethrough**

`[]` (default) | `'none'` | `'single'` | `'double'`

Text strikethrough, specified as one of these values:

- `'none'` — No strikethrough
- `'single'` — Single line
- `'double'` — Double line (for Word reports only)

Setting the `Strike` property adds a corresponding `mlreportgen.dom.Strike` format object to the `Style` property for this document element. Setting the `Strike` property to an empty value removes the object.

### **Style — Text formatting**

array of `mlreportgen.dom.DOCXSection` objects

An array of `mlreportgen.dom.DOCXSection` objects that specifies the format for the text.

### **StyleName — Name of style to apply to text**

character vector

The style specified by `styleName` must be defined in the template used to create the document element to which this text is appended.

Data Types: `char`

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Underline — Type of underline**

[] (default) | 'single' | 'none' | ...

Type of underline, specified as one of the values in this table.

Underline value	Description	Supported Output Types
'single'	Single line	All
'words'	Underline all characters except spaces	Word
'double'	Double line	Word
'thick'	Thick line	Word
'dotted'	Dotted line	Word
'dottedHeavy'	Thick dotted line	Word
'dash'	Dashed line	Word
'dashedHeavy'	Line with heavy dashes	Word
'dashLong'	Line with long dashes	Word
'dashLongHeavy'	Line with heavy long dashes	Word
'dotDash'	Dot-dash line	Word
'dashDotHeavy'	Heavy dash-dot line	Word
'dotDotDash'	Dot-dot-dash line	Word
'dashDotDotHeavy'	Line with heavy dashes with two dots between the dashes	Word
'wave'	Wavy line	Word
'wavyHeavy'	Heavy wavy	Word
'wavyDouble'	Double wavy line	Word
'none'	No underline	All

Setting the Underline property adds a corresponding `mlreportgen.dom.Underline` format object to the Style property. Setting the Underline property to an empty value removes the object.

To specify both the color and the type of the underline, do not set the Underline property. Instead, set the Style property to include an `mlreportgen.dom.Underline` format object that specifies the underline type and color.

**WhiteSpace — How to treat white space in text**

[] (default) | 'normal' | 'nowrap' | ...

How to treat white space in text, specified as one of the values in this table.

Value	Description	Supported Output Types
'normal'	For HTML and PDF, this value removes spaces at the beginning and the end of text. Multiple spaces in the text collapse to a single space.  For Word, this value removes spaces at the beginning and end of text.	All
'nowrap'	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
'pre'	Preserves white space. Text wraps only on line breaks. Acts like the <pre> tag in HTML.	HTML and PDF
'pre-line'	Sequences of white space collapse into a single white space. Text wraps when necessary and on line breaks.	HTML and PDF
'pre-wrap'	Preserves white space. Text wraps when necessary and on line breaks.	HTML and PDF
'preserve'	Same as 'pre'.	All

Setting the `WhiteSpace` property adds a corresponding `mlreportgen.dom.WhiteSpace` format object to the `Style` property. Setting the `WhiteSpace` property to an empty value removes the object.

## Methods

Method	Purpose
<code>append</code>  Use <code>Heading.append</code> similar to how you use <code>Paragraph.append</code> .	Append content to heading.
<code>clone</code>  Use <code>Heading.clone</code> similar to how you use <code>Paragraph.clone</code> .	Copy heading.

## See Also

`mlreportgen.dom.Paragraph` | `mlreportgen.dom.Heading1`

## mlreportgen.dom.Heading1 class

**Package:** mlreportgen.dom

Create Heading1 paragraph

### Description

Create an mlreportgen.dom.Heading1 paragraph object.

### Construction

headingObj = Heading1() creates an empty Heading1 object.

headingObj = Heading1(text) creates the heading containing the specified text.

headingObj = Heading1(text, styleName) creates the heading using the specified style.

headingObj = Heading1(domObj) creates the heading containing the specified DOM object.

### Input Arguments

#### **text — Heading text**

character vector

Heading text, specified as a character vector.

#### **styleName — Style for the heading**

character vector

The name of a style, specified as a character vector. The style must be defined in the template used to create the document that contains this heading.

#### **domObj — DOM object to include in heading**

mlreportgen.dom.ExternalLink object | mlreportgen.dom.Image object |  
mlreportgen.dom.InternalLink object | mlreportgen.dom.LinkTarget object |  
mlreportgen.dom.Text object

DOM object to include in the heading, specified as any of these DOM object types:

- ExternalLink
- Image
- InternalLink
- LinkTarget
- Text

### Properties

#### **BackgroundColor — Background color**

[] (default) | character vector | string scalar

Background color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `BackgroundColor` property adds a corresponding `mlreportgen.dom.BackgroundColor` format object to the `Style` property. Setting the `BackgroundColor` property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

### **Bold — Whether to use bold for text**

[ ] (default) | true or 1 | false or 0

Whether to use bold for text, specified as a numeric or logical 1 (true) or 0 (false). To make text bold, set `Bold` to true or 1.

Setting the `Bold` property adds a corresponding `mlreportgen.dom.Bold` format object to the `Style` property. Setting the `Bold` property to an empty value removes the object.

### **Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

### **Color — Text color**

[ ] (default) | character vector | string scalar

Text color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `Color` property adds a corresponding `mlreportgen.dom.Color` format object to the `Style` property. Setting the `Color` property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

### **CustomAttributes — Custom attributes of this element**

array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. Use custom attributes supported by the output format.

### **FirstLineIndent — Indentation amount for first line of paragraph**

[ ] | character vector | string scalar

Indentation amount for the first line of this paragraph, specified as a character vector or string scalar that consists of number followed by an abbreviation for a unit of measurement. For example, '0.5in' specifies one-half inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters

- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

The indentation amount is the amount by which to indent the first line of this paragraph relative to the succeeding lines. To create a hanging indent where all of the lines are indented except for the first line, use a negative number.

Setting the `FirstLineIndent` property adds a corresponding `mlreportgen.dom.FirstLineIndent` format object to the `Style` property for this document element. Setting the `FirstLineIndent` property to an empty value removes the object.

Example: `'0.5in'`

### **FontFamilyName — Name of font family for text**

`[]` | character vector | string scalar

Name of a font family to use for text, specified as a character vector or string scalar.

Setting the `FontFamilyName` property adds a corresponding `mlreportgen.dom.FontFamily` format object to the `Style` property. Setting the `FontFamilyName` property is the same as setting the `FontName` property of an `mlreportgen.dom.FontFamily` object. Setting the `FontFamilyName` property to an empty value removes the object.

To specify substitutions for the font, do not specify the font family by setting this property. Instead create and add an `mlreportgen.dom.FontFamily` object to the `Style` property.

Example: `'Courier New'`

### **FontSize — Font size for text**

`[]` (default) | character vector | string scalar

Font size to use for text, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, `'12pt'` specifies twelve points. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

Setting the `FontSize` property adds a corresponding `mlreportgen.dom.FontSize` format object to the `Style` property. Setting the `FontSize` property to an empty value removes the object.

Example: `'12pt'`

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Italic — Whether to use italic for text**

[] (default) | true or 1 | false or 0

Whether to use italic for text, specified as a numeric or logical 1 (true) or 0 (false). To make text italic, set `Italic` to true or 1.

Setting the `Italic` property adds a corresponding `mlreportgen.dom.Italic` format object to the `Style` property. Setting the `Italic` property to an empty value removes the object.

**OuterLeftMargin — Left indentation for paragraph**

[] | character vector | string scalar

Left indentation for this paragraph, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '0.5in' specifies one-half inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

The left indentation is the space between the left outer boundary of this paragraph and the left inner boundary of its container. This is equivalent to the left indentation property of a Microsoft Word paragraph.

Setting the `OuterLeftMargin` property adds a corresponding `mlreportGen.dom.OuterMargin` format object to the `Style` property for this document element. Setting the `OuterLeftMargin` property to an empty value removes the object.

To indent a paragraph from both the left and right margin of a page, do not set this property. Instead, create an `mlreportgen.dom.OuterMargin` that specifies the left and right indentations and add the object to the `Style` property of this paragraph.

**OutlineLevel — Outline level of paragraph**

[] (default) | integer

Outline level of the paragraph, specified as an integer. Setting the `OutlineLevel` property causes this paragraph to be included in automatically generated outlines, such as a table of contents. The value specifies the level of the paragraph in the outline. For example, to make a paragraph appear at the top level in an outline, set the `OutlineLevel` property to 1.

Setting the `OutlineLevel` property adds a corresponding `mlreportgen.dom.OutlineLevel` format object to the `Style` property. Setting the `OutlineLevel` property to an empty value removes the object.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Strike — Text strikethrough**

[] (default) | 'none' | 'single' | 'double'

Text strikethrough, specified as one of these values:

- 'none' — No strikethrough
- 'single' — Single line
- 'double' — Double line (for Word reports only)

Setting the `Strike` property adds a corresponding `mreportgen.dom.Strike` format object to the `Style` property for this document element. Setting the `Strike` property to an empty value removes the object.

### Style — Format specification

array of format objects

Format objects that specify the format of a document element.

### StyleName — Style to apply from style sheet

character vector

Name of the style to apply from the style sheet, specified as a character vector.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Underline — Type of underline

[] (default) | 'single' | 'none' | ...

Type of underline, specified as one of the values in this table.

Underline value	Description	Supported Output Types
'single'	Single line	All
'words'	Underline all characters except spaces	Word
'double'	Double line	Word
'thick'	Thick line	Word
'dotted'	Dotted line	Word
'dottedHeavy'	Thick dotted line	Word
'dash'	Dashed line	Word
'dashedHeavy'	Line with heavy dashes	Word
'dashLong'	Line with long dashes	Word
'dashLongHeavy'	Line with heavy long dashes	Word
'dotDash'	Dot-dash line	Word



Underline value	Description	Supported Output Types
'dashDotHeavy'	Heavy dash-dot line	Word
'dotDotDash'	Dot-dot-dash line	Word
'dashDotDotHeavy'	Line with heavy dashes with two dots between the dashes	Word
'wave'	Wavy line	Word
'wavyHeavy'	Heavy wavy	Word
'wavyDouble'	Double wavy line	Word
'none'	No underline	All

Setting the Underline property adds a corresponding `mlreportgen.dom.Underline` format object to the Style property. Setting the Underline property to an empty value removes the object.

To specify both the color and the type of the underline, do not set the Underline property. Instead, set the Style property to include an `mlreportgen.dom.Underline` format object that specifies the underline type and color.

### WhiteSpace – How to treat white space in text

[ ] (default) | 'normal' | 'nowrap' | ...

How to treat white space in text, specified as one of the values in this table.

Value	Description	Supported Output Types
'normal'	For HTML and PDF, this value removes spaces at the beginning and the end of text. Multiple spaces in the text collapse to a single space.  For Word, this value removes spaces at the beginning and end of text.	All
'nowrap'	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
'pre'	Preserves white space. Text wraps only on line breaks. Acts like the <code>&lt;pre&gt;</code> tag in HTML.	HTML and PDF
'pre-line'	Sequences of white space collapse into a single white space. Text wraps when necessary and on line breaks.	HTML and PDF
'pre-wrap'	Preserves white space. Text wraps when necessary and on line breaks.	HTML and PDF
'preserve'	Same as 'pre'.	All

Setting the `WhiteSpace` property adds a corresponding `mlreportgen.dom.WhiteSpace` format object to the `Style` property. Setting the `WhiteSpace` property to an empty value removes the object.

## Methods

Method	Purpose
<code>append</code> Use <code>Heading1.append</code> similar to how you use <code>Paragraph.append</code> .	Append content to heading.
<code>clone</code> Use <code>Heading1.clone</code> similar to how you use <code>Paragraph.clone</code> .	Copy heading.

## Examples

### Create Three Levels of Headings

This example shows how to add three levels of headings, each formatted according to its level. This example inserts three heading objects into a document: a `Heading1`, a `Heading2`, and a `Heading3`.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'docx');
open(d);

title = append(d, Paragraph('Document Title'));
title.Bold = true;
title.FontSize = '28pt';

h1 = append(d, Heading1('Chapter 1'));
h1.Style = {PageBreakBefore(true)};
p1 = append(d, Paragraph('Hello World'));

h2 = append(d, Heading2('Section 1.1'));
p2 = append(d, Paragraph('Text for this section. '));

h3 = append(d, Heading3('My Subsection 1.1.a'));
p3 = append(d, Paragraph('Text for this subsection'));

close(d);
rptview(d.OutputPath);
```

### See Also

`mlreportgen.dom.Paragraph` | `mlreportgen.dom.Heading` | `mlreportgen.dom.Heading2` | `mlreportgen.dom.Heading3` | `mlreportgen.dom.Heading4` | `mlreportgen.dom.Heading5` | `mlreportgen.dom.Heading6`

### Topics

“Report Formatting Approaches” on page 13-17

**Introduced in R2016a**

## mlreportgen.dom.Heading2 class

**Package:** mlreportgen.dom

Create Heading2 paragraph

### Description

Create an mlreportgen.dom.Heading2 paragraph object.

### Construction

headingObj = Heading2() creates an empty Heading2 object.

headingObj = Heading2(text) creates the heading containing the specified text.

headingObj = Heading2(text, styleName) creates the heading using the specified style.

headingObj = Heading2(domObj) creates the heading containing the specified DOM object.

### Input Arguments

#### **text — Heading text**

character vector

Heading text, specified as a character vector.

#### **styleName — Style for the heading**

character vector

The name of a style, specified as a character vector. The style must be defined in the template used to create the document that contains this heading.

#### **domObj — DOM object to include in heading**

mlreportgen.dom.ExternalLink object | mlreportgen.dom.Image object |  
mlreportgen.dom.InternalLink object | mlreportgen.dom.LinkTarget object |  
mlreportgen.dom.Text object

DOM object to include in the heading, specified as any of these DOM object types:

- ExternalLink
- Image
- InternalLink
- LinkTarget
- Text

### Properties

#### **BackgroundColor — Background color**

[] (default) | character vector | string scalar

Background color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `BackgroundColor` property adds a corresponding `mlreportgen.dom.BackgroundColor` format object to the `Style` property. Setting the `BackgroundColor` property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

### **Bold — Whether to use bold for text**

[] (default) | true or 1 | false or 0

Whether to use bold for text, specified as a numeric or logical 1 (true) or 0 (false). To make text bold, set `Bold` to true or 1.

Setting the `Bold` property adds a corresponding `mlreportgen.dom.Bold` format object to the `Style` property. Setting the `Bold` property to an empty value removes the object.

### **Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

### **Color — Text color**

[] (default) | character vector | string scalar

Text color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `Color` property adds a corresponding `mlreportgen.dom.Color` format object to the `Style` property. Setting the `Color` property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

### **CustomAttributes — Custom attributes of this element**

array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. Use custom attributes supported by the output format.

### **FirstLineIndent — Indentation amount for first line of paragraph**

[] | character vector | string scalar

Indentation amount for the first line of this paragraph, specified as a character vector or string scalar that consists of number followed by an abbreviation for a unit of measurement. For example, '0.5in' specifies one-half inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters

- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

The indentation amount is the amount by which to indent the first line of this paragraph relative to the succeeding lines. To create a hanging indent where all of the lines are indented except for the first line, use a negative number.

Setting the `FirstLineIndent` property adds a corresponding `mlreportgen.dom.FirstLineIndent` format object to the `Style` property for this document element. Setting the `FirstLineIndent` property to an empty value removes the object.

Example: `'0.5in'`

### **FontFamilyName — Name of font family for text**

`[]` | character vector | string scalar

Name of a font family to use for text, specified as a character vector or string scalar.

Setting the `FontFamilyName` property adds a corresponding `mlreportgen.dom.FontFamily` format object to the `Style` property. Setting the `FontFamilyName` property is the same as setting the `FontName` property of an `mlreportgen.dom.FontFamily` object. Setting the `FontFamilyName` property to an empty value removes the object.

To specify substitutions for the font, do not specify the font family by setting this property. Instead create and add an `mlreportgen.dom.FontFamily` object to the `Style` property.

Example: `'Courier New'`

### **FontSize — Font size for text**

`[]` (default) | character vector | string scalar

Font size to use for text, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, `'12pt'` specifies twelve points. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

Setting the `FontSize` property adds a corresponding `mlreportgen.dom.FontSize` format object to the `Style` property. Setting the `FontSize` property to an empty value removes the object.

Example: `'12pt'`

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Italic — Whether to use italic for text**

[] (default) | true or 1 | false or 0

Whether to use italic for text, specified as a numeric or logical 1 (true) or 0 (false). To make text italic, set `Italic` to true or 1.

Setting the `Italic` property adds a corresponding `mlreportgen.dom.Italic` format object to the `Style` property. Setting the `Italic` property to an empty value removes the object.

**OuterLeftMargin — Left indentation for paragraph**

[] | character vector | string scalar

Left indentation for this paragraph, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '0.5in' specifies one-half inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

The left indentation is the space between the left outer boundary of this paragraph and the left inner boundary of its container. This is equivalent to the left indentation property of a Microsoft Word paragraph.

Setting the `OuterLeftMargin` property adds a corresponding `mlreportGen.dom.OuterMargin` format object to the `Style` property for this document element. Setting the `OuterLeftMargin` property to an empty value removes the object.

To indent a paragraph from both the left and right margin of a page, do not set this property. Instead, create an `mlreportgen.dom.OuterMargin` that specifies the left and right indentations and add the object to the `Style` property of this paragraph.

**OutlineLevel — Outline level of paragraph**

[] (default) | integer

Outline level of the paragraph, specified as an integer. Setting the `OutlineLevel` property causes this paragraph to be included in automatically generated outlines, such as a table of contents. The value specifies the level of the paragraph in the outline. For example, to make a paragraph appear at the top level in an outline, set the `OutlineLevel` property to 1.

Setting the `OutlineLevel` property adds a corresponding `mlreportgen.dom.OutlineLevel` format object to the `Style` property. Setting the `OutlineLevel` property to an empty value removes the object.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Strike — Text strikethrough**

[] (default) | 'none' | 'single' | 'double'

Text strikethrough, specified as one of these values:

- 'none' — No strikethrough
- 'single' — Single line
- 'double' — Double line (for Word reports only)

Setting the `Strike` property adds a corresponding `mreportgen.dom.Strike` format object to the `Style` property for this document element. Setting the `Strike` property to an empty value removes the object.

### Style — Format specification

array of format objects

Format objects that specify the format of a document element.

### StyleName — Style to apply from style sheet

character vector

Name of the style to apply from the style sheet, specified as a character vector.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Underline — Type of underline

[] (default) | 'single' | 'none' | ...

Type of underline, specified as one of the values in this table.

Underline value	Description	Supported Output Types
'single'	Single line	All
'words'	Underline all characters except spaces	Word
'double'	Double line	Word
'thick'	Thick line	Word
'dotted'	Dotted line	Word
'dottedHeavy'	Thick dotted line	Word
'dash'	Dashed line	Word
'dashedHeavy'	Line with heavy dashes	Word
'dashLong'	Line with long dashes	Word
'dashLongHeavy'	Line with heavy long dashes	Word
'dotDash'	Dot-dash line	Word



Underline value	Description	Supported Output Types
'dashDotHeavy'	Heavy dash-dot line	Word
'dotDotDash'	Dot-dot-dash line	Word
'dashDotDotHeavy'	Line with heavy dashes with two dots between the dashes	Word
'wave'	Wavy line	Word
'wavyHeavy'	Heavy wavy	Word
'wavyDouble'	Double wavy line	Word
'none'	No underline	All

Setting the Underline property adds a corresponding `mlreportgen.dom.Underline` format object to the Style property. Setting the Underline property to an empty value removes the object.

To specify both the color and the type of the underline, do not set the Underline property. Instead, set the Style property to include an `mlreportgen.dom.Underline` format object that specifies the underline type and color.

### WhiteSpace – How to treat white space in text

[ ] (default) | 'normal' | 'nowrap' | ...

How to treat white space in text, specified as one of the values in this table.

Value	Description	Supported Output Types
'normal'	For HTML and PDF, this value removes spaces at the beginning and the end of text. Multiple spaces in the text collapse to a single space.  For Word, this value removes spaces at the beginning and end of text.	All
'nowrap'	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
'pre'	Preserves white space. Text wraps only on line breaks. Acts like the <code>&lt;pre&gt;</code> tag in HTML.	HTML and PDF
'pre-line'	Sequences of white space collapse into a single white space. Text wraps when necessary and on line breaks.	HTML and PDF
'pre-wrap'	Preserves white space. Text wraps when necessary and on line breaks.	HTML and PDF
'preserve'	Same as 'pre'.	All

Setting the `WhiteSpace` property adds a corresponding `mlreportgen.dom.WhiteSpace` format object to the `Style` property. Setting the `WhiteSpace` property to an empty value removes the object.

## Methods

Method	Purpose
<code>append</code> Use <code>Heading2.append</code> similar to how you use <code>Paragraph.append</code> .	Append content to heading.
<code>clone</code> Use <code>Heading2.clone</code> similar to how you use <code>Paragraph.clone</code> .	Copy heading.

## Examples

### Create Three Levels of Headings

This example shows how to add three levels of headings, each formatted according to its level. This example inserts three heading objects into a document: a `Heading1`, a `Heading2`, and a `Heading3`.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'docx');
open(d);

title = append(d, Paragraph('Document Title'));
title.Bold = true;
title.FontSize = '28pt';

h1 = append(d, Heading1('Chapter 1'));
h1.Style = {PageBreakBefore(true)};
p1 = append(d, Paragraph('Hello World'));

h2 = append(d, Heading2('Section 1.1'));
p2 = append(d, Paragraph('Text for this section. '));

h3 = append(d, Heading3('My Subsection 1.1.a'));
p3 = append(d, Paragraph('Text for this subsection'));

close(d);
rptview(d.OutputPath);
```

### See Also

`mlreportgen.dom.Paragraph` | `mlreportgen.dom.Heading` | `mlreportgen.dom.Heading1` | `mlreportgen.dom.Heading3` | `mlreportgen.dom.Heading4` | `mlreportgen.dom.Heading5` | `mlreportgen.dom.Heading6`

### Topics

“Report Formatting Approaches” on page 13-17

**Introduced in R2016a**

## mlreportgen.dom.Heading3 class

**Package:** mlreportgen.dom

Create Heading3 paragraph

### Description

Create an mlreportgen.dom.Heading3 paragraph object.

### Construction

headingObj = Heading3() creates an empty Heading3 object.

headingObj = Heading3(text) creates the heading containing the specified text.

headingObj = Heading3(text, styleName) creates the heading using the specified style.

headingObj = Heading3(domObj) creates the heading containing the specified DOM object.

### Input Arguments

#### **text — Heading text**

character vector

Heading text, specified as a character vector.

#### **styleName — Style for the heading**

character vector

The name of a style, specified as a character vector. The style must be defined in the template used to create the document that contains this heading.

#### **domObj — DOM object to include in heading**

mlreportgen.dom.ExternalLink object | mlreportgen.dom.Image object |  
mlreportgen.dom.InternalLink object | mlreportgen.dom.LinkTarget object |  
mlreportgen.dom.Text object

DOM object to include in the heading, specified as any of these DOM object types:

- ExternalLink
- Image
- InternalLink
- LinkTarget
- Text

### Properties

#### **BackgroundColor — Background color**

[] (default) | character vector | string scalar

Background color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `BackgroundColor` property adds a corresponding `mlreportgen.dom.BackgroundColor` format object to the `Style` property. Setting the `BackgroundColor` property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

### **Bold — Whether to use bold for text**

[ ] (default) | true or 1 | false or 0

Whether to use bold for text, specified as a numeric or logical 1 (true) or 0 (false). To make text bold, set `Bold` to true or 1.

Setting the `Bold` property adds a corresponding `mlreportgen.dom.Bold` format object to the `Style` property. Setting the `Bold` property to an empty value removes the object.

### **Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

### **Color — Text color**

[ ] (default) | character vector | string scalar

Text color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `Color` property adds a corresponding `mlreportgen.dom.Color` format object to the `Style` property. Setting the `Color` property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

### **CustomAttributes — Custom attributes of this element**

array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. Use custom attributes supported by the output format.

### **FirstLineIndent — Indentation amount for first line of paragraph**

[ ] | character vector | string scalar

Indentation amount for the first line of this paragraph, specified as a character vector or string scalar that consists of number followed by an abbreviation for a unit of measurement. For example, '0.5in' specifies one-half inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters

- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

The indentation amount is the amount by which to indent the first line of this paragraph relative to the succeeding lines. To create a hanging indent where all of the lines are indented except for the first line, use a negative number.

Setting the `FirstLineIndent` property adds a corresponding `mlreportgen.dom.FirstLineIndent` format object to the `Style` property for this document element. Setting the `FirstLineIndent` property to an empty value removes the object.

Example: `'0.5in'`

### **FontFamilyName — Name of font family for text**

`[]` | character vector | string scalar

Name of a font family to use for text, specified as a character vector or string scalar.

Setting the `FontFamilyName` property adds a corresponding `mlreportgen.dom.FontFamily` format object to the `Style` property. Setting the `FontFamilyName` property is the same as setting the `FontName` property of an `mlreportgen.dom.FontFamily` object. Setting the `FontFamilyName` property to an empty value removes the object.

To specify substitutions for the font, do not specify the font family by setting this property. Instead create and add an `mlreportgen.dom.FontFamily` object to the `Style` property.

Example: `'Courier New'`

### **FontSize — Font size for text**

`[]` (default) | character vector | string scalar

Font size to use for text, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, `'12pt'` specifies twelve points. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

Setting the `FontSize` property adds a corresponding `mlreportgen.dom.FontSize` format object to the `Style` property. Setting the `FontSize` property to an empty value removes the object.

Example: `'12pt'`

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Italic — Whether to use italic for text**

[] (default) | true or 1 | false or 0

Whether to use italic for text, specified as a numeric or logical 1 (true) or 0 (false). To make text italic, set `Italic` to true or 1.

Setting the `Italic` property adds a corresponding `mlreportgen.dom.Italic` format object to the `Style` property. Setting the `Italic` property to an empty value removes the object.

**OuterLeftMargin — Left indentation for paragraph**

[] | character vector | string scalar

Left indentation for this paragraph, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '0.5in' specifies one-half inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

The left indentation is the space between the left outer boundary of this paragraph and the left inner boundary of its container. This is equivalent to the left indentation property of a Microsoft Word paragraph.

Setting the `OuterLeftMargin` property adds a corresponding `mlreportGen.dom.OuterMargin` format object to the `Style` property for this document element. Setting the `OuterLeftMargin` property to an empty value removes the object.

To indent a paragraph from both the left and right margin of a page, do not set this property. Instead, create an `mlreportgen.dom.OuterMargin` that specifies the left and right indentations and add the object to the `Style` property of this paragraph.

**OutlineLevel — Outline level of paragraph**

[] (default) | integer

Outline level of the paragraph, specified as an integer. Setting the `OutlineLevel` property causes this paragraph to be included in automatically generated outlines, such as a table of contents. The value specifies the level of the paragraph in the outline. For example, to make a paragraph appear at the top level in an outline, set the `OutlineLevel` property to 1.

Setting the `OutlineLevel` property adds a corresponding `mlreportgen.dom.OutlineLevel` format object to the `Style` property. Setting the `OutlineLevel` property to an empty value removes the object.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Strike — Text strikethrough**

[] (default) | 'none' | 'single' | 'double'

Text strikethrough, specified as one of these values:

- 'none' — No strikethrough
- 'single' — Single line
- 'double' — Double line (for Word reports only)

Setting the `Strike` property adds a corresponding `mreportgen.dom.Strike` format object to the `Style` property for this document element. Setting the `Strike` property to an empty value removes the object.

### Style — Format specification

array of format objects

Format objects that specify the format of a document element.

### StyleName — Style to apply from style sheet

character vector

Name of the style to apply from the style sheet, specified as a character vector.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Underline — Type of underline

[] (default) | 'single' | 'none' | ...

Type of underline, specified as one of the values in this table.

Underline value	Description	Supported Output Types
'single'	Single line	All
'words'	Underline all characters except spaces	Word
'double'	Double line	Word
'thick'	Thick line	Word
'dotted'	Dotted line	Word
'dottedHeavy'	Thick dotted line	Word
'dash'	Dashed line	Word
'dashedHeavy'	Line with heavy dashes	Word
'dashLong'	Line with long dashes	Word
'dashLongHeavy'	Line with heavy long dashes	Word
'dotDash'	Dot-dash line	Word



Underline value	Description	Supported Output Types
'dashDotHeavy'	Heavy dash-dot line	Word
'dotDotDash'	Dot-dot-dash line	Word
'dashDotDotHeavy'	Line with heavy dashes with two dots between the dashes	Word
'wave'	Wavy line	Word
'wavyHeavy'	Heavy wavy	Word
'wavyDouble'	Double wavy line	Word
'none'	No underline	All

Setting the Underline property adds a corresponding `mlreportgen.dom.Underline` format object to the Style property. Setting the Underline property to an empty value removes the object.

To specify both the color and the type of the underline, do not set the Underline property. Instead, set the Style property to include an `mlreportgen.dom.Underline` format object that specifies the underline type and color.

### WhiteSpace – How to treat white space in text

[ ] (default) | 'normal' | 'nowrap' | ...

How to treat white space in text, specified as one of the values in this table.

Value	Description	Supported Output Types
'normal'	For HTML and PDF, this value removes spaces at the beginning and the end of text. Multiple spaces in the text collapse to a single space.  For Word, this value removes spaces at the beginning and end of text.	All
'nowrap'	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
'pre'	Preserves white space. Text wraps only on line breaks. Acts like the <code>&lt;pre&gt;</code> tag in HTML.	HTML and PDF
'pre-line'	Sequences of white space collapse into a single white space. Text wraps when necessary and on line breaks.	HTML and PDF
'pre-wrap'	Preserves white space. Text wraps when necessary and on line breaks.	HTML and PDF
'preserve'	Same as 'pre'.	All

Setting the `WhiteSpace` property adds a corresponding `mlreportgen.dom.WhiteSpace` format object to the `Style` property. Setting the `WhiteSpace` property to an empty value removes the object.

## Methods

Methods

Method	Purpose
<b>append</b> Use <code>Heading3.append</code> similar to how you use <code>Paragraph.append</code> .	Append content to heading.
<b>clone</b> Use <code>Heading3.clone</code> similar to how you use <code>Paragraph.clone</code> .	Copy heading.

## Examples

### Create Three Levels of Headings

This example shows how to add three levels of headings, each formatted according to its level. This example inserts three heading objects into a document: a `Heading1`, a `Heading2`, and a `Heading3`.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'docx');
open(d);

title = append(d, Paragraph('Document Title'));
title.Bold = true;
title.FontSize = '28pt';

h1 = append(d, Heading1('Chapter 1'));
h1.Style = {PageBreakBefore(true)};
p1 = append(d, Paragraph('Hello World'));

h2 = append(d, Heading2('Section 1.1'));
p2 = append(d, Paragraph('Text for this section. '));

h3 = append(d, Heading3('My Subsection 1.1.a'));
p3 = append(d, Paragraph('Text for this subsection'));

close(d);
rptview(d.OutputPath);
```

### See Also

`mlreportgen.dom.Paragraph` | `mlreportgen.dom.Heading` | `mlreportgen.dom.Heading1` | `mlreportgen.dom.Heading2` | `mlreportgen.dom.Heading4` | `mlreportgen.dom.Heading5` | `mlreportgen.dom.Heading6`

**Topics**

“Report Formatting Approaches” on page 13-17

**Introduced in R2016a**

## mlreportgen.dom.Heading4 class

**Package:** mlreportgen.dom

Create Heading4 paragraph

### Description

Create an mlreportgen.dom.Heading4 paragraph object.

### Construction

headingObj = Heading4() creates an empty Heading4 object.

headingObj = Heading4(text) creates the heading containing the specified text.

headingObj = Heading4(text, styleName) creates the heading using the specified style.

headingObj = Heading4(domObj) creates the heading containing the specified DOM object.

### Input Arguments

#### **text — Heading text**

character vector

Heading text, specified as a character vector.

#### **styleName — Style for the heading**

character vector

The name of a style, specified as a character vector. The style must be defined in the template used to create the document that contains this heading.

#### **domObj — DOM object to include in heading**

mlreportgen.dom.ExternalLink object | mlreportgen.dom.Image object |  
mlreportgen.dom.InternalLink object | mlreportgen.dom.LinkTarget object |  
mlreportgen.dom.Text object

DOM object to include in the heading, specified as any of these DOM object types:

- ExternalLink
- Image
- InternalLink
- LinkTarget
- Text

### Properties

#### **BackgroundColor — Background color**

[] (default) | character vector | string scalar

Background color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `BackgroundColor` property adds a corresponding `mlreportgen.dom.BackgroundColor` format object to the `Style` property. Setting the `BackgroundColor` property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

### **Bold — Whether to use bold for text**

[] (default) | true or 1 | false or 0

Whether to use bold for text, specified as a numeric or logical 1 (true) or 0 (false). To make text bold, set `Bold` to true or 1.

Setting the `Bold` property adds a corresponding `mlreportgen.dom.Bold` format object to the `Style` property. Setting the `Bold` property to an empty value removes the object.

### **Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

### **Color — Text color**

[] (default) | character vector | string scalar

Text color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `Color` property adds a corresponding `mlreportgen.dom.Color` format object to the `Style` property. Setting the `Color` property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

### **CustomAttributes — Custom attributes of this element**

array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. Use custom attributes supported by the output format.

### **FirstLineIndent — Indentation amount for first line of paragraph**

[] | character vector | string scalar

Indentation amount for the first line of this paragraph, specified as a character vector or string scalar that consists of number followed by an abbreviation for a unit of measurement. For example, '0.5in' specifies one-half inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters

- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

The indentation amount is the amount by which to indent the first line of this paragraph relative to the succeeding lines. To create a hanging indent where all of the lines are indented except for the first line, use a negative number.

Setting the `FirstLineIndent` property adds a corresponding `mlreportgen.dom.FirstLineIndent` format object to the `Style` property for this document element. Setting the `FirstLineIndent` property to an empty value removes the object.

Example: `'0.5in'`

### **FontFamilyName — Name of font family for text**

`[]` | character vector | string scalar

Name of a font family to use for text, specified as a character vector or string scalar.

Setting the `FontFamilyName` property adds a corresponding `mlreportgen.dom.FontFamily` format object to the `Style` property. Setting the `FontFamilyName` property is the same as setting the `FontName` property of an `mlreportgen.dom.FontFamily` object. Setting the `FontFamilyName` property to an empty value removes the object.

To specify substitutions for the font, do not specify the font family by setting this property. Instead create and add an `mlreportgen.dom.FontFamily` object to the `Style` property.

Example: `'Courier New'`

### **FontSize — Font size for text**

`[]` (default) | character vector | string scalar

Font size to use for text, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, `'12pt'` specifies twelve points. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

Setting the `FontSize` property adds a corresponding `mlreportgen.dom.FontSize` format object to the `Style` property. Setting the `FontSize` property to an empty value removes the object.

Example: `'12pt'`

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Italic — Whether to use italic for text**

[ ] (default) | true or 1 | false or 0

Whether to use italic for text, specified as a numeric or logical 1 (true) or 0 (false). To make text italic, set `Italic` to true or 1.

Setting the `Italic` property adds a corresponding `mlreportgen.dom.Italic` format object to the `Style` property. Setting the `Italic` property to an empty value removes the object.

**OuterLeftMargin — Left indentation for paragraph**

[ ] | character vector | string scalar

Left indentation for this paragraph, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '0.5in' specifies one-half inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

The left indentation is the space between the left outer boundary of this paragraph and the left inner boundary of its container. This is equivalent to the left indentation property of a Microsoft Word paragraph.

Setting the `OuterLeftMargin` property adds a corresponding `mlreportGen.dom.OuterMargin` format object to the `Style` property for this document element. Setting the `OuterLeftMargin` property to an empty value removes the object.

To indent a paragraph from both the left and right margin of a page, do not set this property. Instead, create an `mlreportgen.dom.OuterMargin` that specifies the left and right indentations and add the object to the `Style` property of this paragraph.

**OutlineLevel — Outline level of paragraph**

[ ] (default) | integer

Outline level of the paragraph, specified as an integer. Setting the `OutlineLevel` property causes this paragraph to be included in automatically generated outlines, such as a table of contents. The value specifies the level of the paragraph in the outline. For example, to make a paragraph appear at the top level in an outline, set the `OutlineLevel` property to 1.

Setting the `OutlineLevel` property adds a corresponding `mlreportgen.dom.OutlineLevel` format object to the `Style` property. Setting the `OutlineLevel` property to an empty value removes the object.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Strike — Text strikethrough**

[ ] (default) | 'none' | 'single' | 'double'

Text strikethrough, specified as one of these values:

- 'none' — No strikethrough
- 'single' — Single line
- 'double' — Double line (for Word reports only)

Setting the `Strike` property adds a corresponding `mreportgen.dom.Strike` format object to the `Style` property for this document element. Setting the `Strike` property to an empty value removes the object.

### Style — Format specification

array of format objects

Format objects that specify the format of a document element.

### StyleName — Style to apply from style sheet

character vector

Name of the style to apply from the style sheet, specified as a character vector.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Underline — Type of underline

[] (default) | 'single' | 'none' | ...

Type of underline, specified as one of the values in this table.

Underline value	Description	Supported Output Types
'single'	Single line	All
'words'	Underline all characters except spaces	Word
'double'	Double line	Word
'thick'	Thick line	Word
'dotted'	Dotted line	Word
'dottedHeavy'	Thick dotted line	Word
'dash'	Dashed line	Word
'dashedHeavy'	Line with heavy dashes	Word
'dashLong'	Line with long dashes	Word
'dashLongHeavy'	Line with heavy long dashes	Word
'dotDash'	Dot-dash line	Word



Underline value	Description	Supported Output Types
'dashDotHeavy'	Heavy dash-dot line	Word
'dotDotDash'	Dot-dot-dash line	Word
'dashDotDotHeavy'	Line with heavy dashes with two dots between the dashes	Word
'wave'	Wavy line	Word
'wavyHeavy'	Heavy wavy	Word
'wavyDouble'	Double wavy line	Word
'none'	No underline	All

Setting the Underline property adds a corresponding `mlreportgen.dom.Underline` format object to the Style property. Setting the Underline property to an empty value removes the object.

To specify both the color and the type of the underline, do not set the Underline property. Instead, set the Style property to include an `mlreportgen.dom.Underline` format object that specifies the underline type and color.

### WhiteSpace – How to treat white space in text

[ ] (default) | 'normal' | 'nowrap' | ...

How to treat white space in text, specified as one of the values in this table.

Value	Description	Supported Output Types
'normal'	For HTML and PDF, this value removes spaces at the beginning and the end of text. Multiple spaces in the text collapse to a single space.  For Word, this value removes spaces at the beginning and end of text.	All
'nowrap'	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
'pre'	Preserves white space. Text wraps only on line breaks. Acts like the <code>&lt;pre&gt;</code> tag in HTML.	HTML and PDF
'pre-line'	Sequences of white space collapse into a single white space. Text wraps when necessary and on line breaks.	HTML and PDF
'pre-wrap'	Preserves white space. Text wraps when necessary and on line breaks.	HTML and PDF
'preserve'	Same as 'pre'.	All

Setting the `WhiteSpace` property adds a corresponding `mlreportgen.dom.WhiteSpace` format object to the `Style` property. Setting the `WhiteSpace` property to an empty value removes the object.

## Methods

Method	Purpose
<code>append</code> Use <code>Heading4.append</code> similar to how you use <code>Paragraph.append</code> .	Append content to heading.
<code>clone</code> Use <code>Heading4.clone</code> similar to how you use <code>Paragraph.clone</code> .	Copy heading.

## Examples

### Create Three Levels of Headings

This example shows how to add three levels of headings, each formatted according to its level. This example inserts three heading objects into a document: a `Heading1`, a `Heading2`, and a `Heading3`.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'docx');
open(d);

title = append(d, Paragraph('Document Title'));
title.Bold = true;
title.FontSize = '28pt';

h1 = append(d, Heading1('Chapter 1'));
h1.Style = {PageBreakBefore(true)};
p1 = append(d, Paragraph('Hello World'));

h2 = append(d, Heading2('Section 1.1'));
p2 = append(d, Paragraph('Text for this section. '));

h3 = append(d, Heading3('My Subsection 1.1.a'));
p3 = append(d, Paragraph('Text for this subsection'));

close(d);
rptview(d.OutputPath);
```

### See Also

`mlreportgen.dom.Paragraph` | `mlreportgen.dom.Heading` | `mlreportgen.dom.Heading1` | `mlreportgen.dom.Heading2` | `mlreportgen.dom.Heading3` | `mlreportgen.dom.Heading5` | `mlreportgen.dom.Heading6`

### Topics

“Report Formatting Approaches” on page 13-17

**Introduced in R2016a**

## mlreportgen.dom.Heading5 class

**Package:** mlreportgen.dom

Create Heading5 paragraph

### Description

Create an mlreportgen.dom.Heading5 paragraph object.

### Construction

headingObj = Heading5() creates an empty Heading5 object.

headingObj = Heading5(text) creates the heading containing the specified text.

headingObj = Heading5(text, styleName) creates the heading using the specified style.

headingObj = Heading5(domObj) creates the heading containing the specified DOM object.

### Input Arguments

#### **text — Heading text**

character vector

Heading text, specified as a character vector.

#### **styleName — Style for the heading**

character vector

The name of a style, specified as a character vector. The style must be defined in the template used to create the document that contains this heading.

#### **domObj — DOM object to include in heading**

mlreportgen.dom.ExternalLink object | mlreportgen.dom.Image object |  
mlreportgen.dom.InternalLink object | mlreportgen.dom.LinkTarget object |  
mlreportgen.dom.Text object

DOM object to include in the heading, specified as any of these DOM object types:

- ExternalLink
- Image
- InternalLink
- LinkTarget
- Text

### Properties

#### **BackgroundColor — Background color**

[] (default) | character vector | string scalar

Background color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `BackgroundColor` property adds a corresponding `mlreportgen.dom.BackgroundColor` format object to the `Style` property. Setting the `BackgroundColor` property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

### **Bold — Whether to use bold for text**

[ ] (default) | true or 1 | false or 0

Whether to use bold for text, specified as a numeric or logical 1 (true) or 0 (false). To make text bold, set `Bold` to true or 1.

Setting the `Bold` property adds a corresponding `mlreportgen.dom.Bold` format object to the `Style` property. Setting the `Bold` property to an empty value removes the object.

### **Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

### **Color — Text color**

[ ] (default) | character vector | string scalar

Text color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `Color` property adds a corresponding `mlreportgen.dom.Color` format object to the `Style` property. Setting the `Color` property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

### **CustomAttributes — Custom attributes of this element**

array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. Use custom attributes supported by the output format.

### **FirstLineIndent — Indentation amount for first line of paragraph**

[ ] | character vector | string scalar

Indentation amount for the first line of this paragraph, specified as a character vector or string scalar that consists of number followed by an abbreviation for a unit of measurement. For example, '0.5in' specifies one-half inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters

- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

The indentation amount is the amount by which to indent the first line of this paragraph relative to the succeeding lines. To create a hanging indent where all of the lines are indented except for the first line, use a negative number.

Setting the `FirstLineIndent` property adds a corresponding `mlreportgen.dom.FirstLineIndent` format object to the `Style` property for this document element. Setting the `FirstLineIndent` property to an empty value removes the object.

Example: `'0.5in'`

### **FontFamilyName — Name of font family for text**

`[]` | character vector | string scalar

Name of a font family to use for text, specified as a character vector or string scalar.

Setting the `FontFamilyName` property adds a corresponding `mlreportgen.dom.FontFamily` format object to the `Style` property. Setting the `FontFamilyName` property is the same as setting the `FontName` property of an `mlreportgen.dom.FontFamily` object. Setting the `FontFamilyName` property to an empty value removes the object.

To specify substitutions for the font, do not specify the font family by setting this property. Instead create and add an `mlreportgen.dom.FontFamily` object to the `Style` property.

Example: `'Courier New'`

### **FontSize — Font size for text**

`[]` (default) | character vector | string scalar

Font size to use for text, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, `'12pt'` specifies twelve points. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

Setting the `FontSize` property adds a corresponding `mlreportgen.dom.FontSize` format object to the `Style` property. Setting the `FontSize` property to an empty value removes the object.

Example: `'12pt'`

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Italic — Whether to use italic for text**

[] (default) | true or 1 | false or 0

Whether to use italic for text, specified as a numeric or logical 1 (true) or 0 (false). To make text italic, set `Italic` to true or 1.

Setting the `Italic` property adds a corresponding `mlreportgen.dom.Italic` format object to the `Style` property. Setting the `Italic` property to an empty value removes the object.

**OuterLeftMargin — Left indentation for paragraph**

[] | character vector | string scalar

Left indentation for this paragraph, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '0.5in' specifies one-half inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

The left indentation is the space between the left outer boundary of this paragraph and the left inner boundary of its container. This is equivalent to the left indentation property of a Microsoft Word paragraph.

Setting the `OuterLeftMargin` property adds a corresponding `mlreportGen.dom.OuterMargin` format object to the `Style` property for this document element. Setting the `OuterLeftMargin` property to an empty value removes the object.

To indent a paragraph from both the left and right margin of a page, do not set this property. Instead, create an `mlreportgen.dom.OuterMargin` that specifies the left and right indentations and add the object to the `Style` property of this paragraph.

**OutlineLevel — Outline level of paragraph**

[] (default) | integer

Outline level of the paragraph, specified as an integer. Setting the `OutlineLevel` property causes this paragraph to be included in automatically generated outlines, such as a table of contents. The value specifies the level of the paragraph in the outline. For example, to make a paragraph appear at the top level in an outline, set the `OutlineLevel` property to 1.

Setting the `OutlineLevel` property adds a corresponding `mlreportgen.dom.OutlineLevel` format object to the `Style` property. Setting the `OutlineLevel` property to an empty value removes the object.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Strike — Text strikethrough**

[] (default) | 'none' | 'single' | 'double'

Text strikethrough, specified as one of these values:

- 'none' — No strikethrough
- 'single' — Single line
- 'double' — Double line (for Word reports only)

Setting the `Strike` property adds a corresponding `mreportgen.dom.Strike` format object to the `Style` property for this document element. Setting the `Strike` property to an empty value removes the object.

### Style — Format specification

array of format objects

Format objects that specify the format of a document element.

### StyleName — Style to apply from style sheet

character vector

Name of the style to apply from the style sheet, specified as a character vector.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Underline — Type of underline

[] (default) | 'single' | 'none' | ...

Type of underline, specified as one of the values in this table.

Underline value	Description	Supported Output Types
'single'	Single line	All
'words'	Underline all characters except spaces	Word
'double'	Double line	Word
'thick'	Thick line	Word
'dotted'	Dotted line	Word
'dottedHeavy'	Thick dotted line	Word
'dash'	Dashed line	Word
'dashedHeavy'	Line with heavy dashes	Word
'dashLong'	Line with long dashes	Word
'dashLongHeavy'	Line with heavy long dashes	Word
'dotDash'	Dot-dash line	Word



Underline value	Description	Supported Output Types
'dashDotHeavy'	Heavy dash-dot line	Word
'dotDotDash'	Dot-dot-dash line	Word
'dashDotDotHeavy'	Line with heavy dashes with two dots between the dashes	Word
'wave'	Wavy line	Word
'wavyHeavy'	Heavy wavy	Word
'wavyDouble'	Double wavy line	Word
'none'	No underline	All

Setting the Underline property adds a corresponding `mlreportgen.dom.Underline` format object to the Style property. Setting the Underline property to an empty value removes the object.

To specify both the color and the type of the underline, do not set the Underline property. Instead, set the Style property to include an `mlreportgen.dom.Underline` format object that specifies the underline type and color.

### WhiteSpace – How to treat white space in text

[ ] (default) | 'normal' | 'nowrap' | ...

How to treat white space in text, specified as one of the values in this table.

Value	Description	Supported Output Types
'normal'	For HTML and PDF, this value removes spaces at the beginning and the end of text. Multiple spaces in the text collapse to a single space.  For Word, this value removes spaces at the beginning and end of text.	All
'nowrap'	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
'pre'	Preserves white space. Text wraps only on line breaks. Acts like the <code>&lt;pre&gt;</code> tag in HTML.	HTML and PDF
'pre-line'	Sequences of white space collapse into a single white space. Text wraps when necessary and on line breaks.	HTML and PDF
'pre-wrap'	Preserves white space. Text wraps when necessary and on line breaks.	HTML and PDF
'preserve'	Same as 'pre'.	All

Setting the `WhiteSpace` property adds a corresponding `mlreportgen.dom.WhiteSpace` format object to the `Style` property. Setting the `WhiteSpace` property to an empty value removes the object.

## Methods

Method	Purpose
<code>append</code> Use <code>Heading5.append</code> similar to how you use <code>Paragraph.append</code> .	Append content to heading.
<code>clone</code> Use <code>Heading5.clone</code> similar to how you use <code>Paragraph.clone</code> .	Copy heading.

## Examples

### Create Three Levels of Headings

This example shows how to add three levels of headings, each formatted according to its level. This example inserts three heading objects into a document: a `Heading1`, a `Heading2`, and a `Heading3`.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'docx');
open(d);

title = append(d, Paragraph('Document Title'));
title.Bold = true;
title.FontSize = '28pt';

h1 = append(d, Heading1('Chapter 1'));
h1.Style = {PageBreakBefore(true)};
p1 = append(d, Paragraph('Hello World'));

h2 = append(d, Heading2('Section 1.1'));
p2 = append(d, Paragraph('Text for this section. '));

h3 = append(d, Heading3('My Subsection 1.1.a'));
p3 = append(d, Paragraph('Text for this subsection'));

close(d);
rptview(d.OutputPath);
```

### See Also

`mlreportgen.dom.Paragraph` | `mlreportgen.dom.Heading` | `mlreportgen.dom.Heading1` | `mlreportgen.dom.Heading2` | `mlreportgen.dom.Heading3` | `mlreportgen.dom.Heading4` | `mlreportgen.dom.Heading6`

### Topics

“Report Formatting Approaches” on page 13-17

**Introduced in R2016a**

## mlreportgen.dom.Heading6 class

**Package:** mlreportgen.dom

Create Heading6 paragraph

### Description

Create an mlreportgen.dom.Heading6 paragraph object.

### Construction

headingObj = Heading6() creates an empty Heading6 object.

headingObj = Heading6(text) creates the heading containing the specified text.

headingObj = Heading6(text, styleName) creates the heading using the specified style.

headingObj = Heading6(domObj) creates the heading containing the specified DOM object.

### Input Arguments

#### **text — Heading text**

character vector

Heading text, specified as a character vector.

#### **styleName — Style for the heading**

character vector

The name of a style, specified as a character vector. The style must be defined in the template used to create the document that contains this heading.

#### **domObj — DOM object to include in heading**

mlreportgen.dom.ExternalLink object | mlreportgen.dom.Image object |  
mlreportgen.dom.InternalLink object | mlreportgen.dom.LinkTarget object |  
mlreportgen.dom.Text object

DOM object to include in the heading, specified as any of these DOM object types:

- ExternalLink
- Image
- InternalLink
- LinkTarget
- Text

### Properties

#### **BackgroundColor — Background color**

[] (default) | character vector | string scalar

Background color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `BackgroundColor` property adds a corresponding `mlreportgen.dom.BackgroundColor` format object to the `Style` property. Setting the `BackgroundColor` property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

### **Bold — Whether to use bold for text**

[ ] (default) | true or 1 | false or 0

Whether to use bold for text, specified as a numeric or logical 1 (true) or 0 (false). To make text bold, set `Bold` to true or 1.

Setting the `Bold` property adds a corresponding `mlreportgen.dom.Bold` format object to the `Style` property. Setting the `Bold` property to an empty value removes the object.

### **Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

### **Color — Text color**

[ ] (default) | character vector | string scalar

Text color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `Color` property adds a corresponding `mlreportgen.dom.Color` format object to the `Style` property. Setting the `Color` property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

### **CustomAttributes — Custom attributes of this element**

array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. Use custom attributes supported by the output format.

### **FirstLineIndent — Indentation amount for first line of paragraph**

[ ] | character vector | string scalar

Indentation amount for the first line of this paragraph, specified as a character vector or string scalar that consists of number followed by an abbreviation for a unit of measurement. For example, '0.5in' specifies one-half inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters

- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

The indentation amount is the amount by which to indent the first line of this paragraph relative to the succeeding lines. To create a hanging indent where all of the lines are indented except for the first line, use a negative number.

Setting the `FirstLineIndent` property adds a corresponding `mlreportgen.dom.FirstLineIndent` format object to the `Style` property for this document element. Setting the `FirstLineIndent` property to an empty value removes the object.

Example: `'0.5in'`

### **FontFamilyName — Name of font family for text**

`[]` | character vector | string scalar

Name of a font family to use for text, specified as a character vector or string scalar.

Setting the `FontFamilyName` property adds a corresponding `mlreportgen.dom.FontFamily` format object to the `Style` property. Setting the `FontFamilyName` property is the same as setting the `FontName` property of an `mlreportgen.dom.FontFamily` object. Setting the `FontFamilyName` property to an empty value removes the object.

To specify substitutions for the font, do not specify the font family by setting this property. Instead create and add an `mlreportgen.dom.FontFamily` object to the `Style` property.

Example: `'Courier New'`

### **FontSize — Font size for text**

`[]` (default) | character vector | string scalar

Font size to use for text, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, `'12pt'` specifies twelve points. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

Setting the `FontSize` property adds a corresponding `mlreportgen.dom.FontSize` format object to the `Style` property. Setting the `FontSize` property to an empty value removes the object.

Example: `'12pt'`

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Italic — Whether to use italic for text**

[ ] (default) | true or 1 | false or 0

Whether to use italic for text, specified as a numeric or logical 1 (true) or 0 (false). To make text italic, set `Italic` to true or 1.

Setting the `Italic` property adds a corresponding `mlreportgen.dom.Italic` format object to the `Style` property. Setting the `Italic` property to an empty value removes the object.

**OuterLeftMargin — Left indentation for paragraph**

[ ] | character vector | string scalar

Left indentation for this paragraph, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '0.5in' specifies one-half inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

The left indentation is the space between the left outer boundary of this paragraph and the left inner boundary of its container. This is equivalent to the left indentation property of a Microsoft Word paragraph.

Setting the `OuterLeftMargin` property adds a corresponding `mlreportGen.dom.OuterMargin` format object to the `Style` property for this document element. Setting the `OuterLeftMargin` property to an empty value removes the object.

To indent a paragraph from both the left and right margin of a page, do not set this property. Instead, create an `mlreportgen.dom.OuterMargin` that specifies the left and right indentations and add the object to the `Style` property of this paragraph.

**OutlineLevel — Outline level of paragraph**

[ ] (default) | integer

Outline level of the paragraph, specified as an integer. Setting the `OutlineLevel` property causes this paragraph to be included in automatically generated outlines, such as a table of contents. The value specifies the level of the paragraph in the outline. For example, to make a paragraph appear at the top level in an outline, set the `OutlineLevel` property to 1.

Setting the `OutlineLevel` property adds a corresponding `mlreportgen.dom.OutlineLevel` format object to the `Style` property. Setting the `OutlineLevel` property to an empty value removes the object.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Strike — Text strikethrough**

[ ] (default) | 'none' | 'single' | 'double'

Text strikethrough, specified as one of these values:

- 'none' — No strikethrough
- 'single' — Single line
- 'double' — Double line (for Word reports only)

Setting the `Strike` property adds a corresponding `mreportgen.dom.Strike` format object to the `Style` property for this document element. Setting the `Strike` property to an empty value removes the object.

### Style — Format specification

array of format objects

Format objects that specify the format of a document element.

### StyleName — Style to apply from style sheet

character vector

Name of the style to apply from the style sheet, specified as a character vector.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Underline — Type of underline

[] (default) | 'single' | 'none' | ...

Type of underline, specified as one of the values in this table.

Underline value	Description	Supported Output Types
'single'	Single line	All
'words'	Underline all characters except spaces	Word
'double'	Double line	Word
'thick'	Thick line	Word
'dotted'	Dotted line	Word
'dottedHeavy'	Thick dotted line	Word
'dash'	Dashed line	Word
'dashedHeavy'	Line with heavy dashes	Word
'dashLong'	Line with long dashes	Word
'dashLongHeavy'	Line with heavy long dashes	Word
'dotDash'	Dot-dash line	Word



Underline value	Description	Supported Output Types
'dashDotHeavy'	Heavy dash-dot line	Word
'dotDotDash'	Dot-dot-dash line	Word
'dashDotDotHeavy'	Line with heavy dashes with two dots between the dashes	Word
'wave'	Wavy line	Word
'wavyHeavy'	Heavy wavy	Word
'wavyDouble'	Double wavy line	Word
'none'	No underline	All

Setting the Underline property adds a corresponding `mlreportgen.dom.Underline` format object to the Style property. Setting the Underline property to an empty value removes the object.

To specify both the color and the type of the underline, do not set the Underline property. Instead, set the Style property to include an `mlreportgen.dom.Underline` format object that specifies the underline type and color.

### WhiteSpace – How to treat white space in text

[ ] (default) | 'normal' | 'nowrap' | ...

How to treat white space in text, specified as one of the values in this table.

Value	Description	Supported Output Types
'normal'	For HTML and PDF, this value removes spaces at the beginning and the end of text. Multiple spaces in the text collapse to a single space.  For Word, this value removes spaces at the beginning and end of text.	All
'nowrap'	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
'pre'	Preserves white space. Text wraps only on line breaks. Acts like the <code>&lt;pre&gt;</code> tag in HTML.	HTML and PDF
'pre-line'	Sequences of white space collapse into a single white space. Text wraps when necessary and on line breaks.	HTML and PDF
'pre-wrap'	Preserves white space. Text wraps when necessary and on line breaks.	HTML and PDF
'preserve'	Same as 'pre'.	All

Setting the `WhiteSpace` property adds a corresponding `mlreportgen.dom.WhiteSpace` format object to the `Style` property. Setting the `WhiteSpace` property to an empty value removes the object.

## Methods

Method	Purpose
<code>append</code> Use <code>Heading6.append</code> similar to how you use <code>Paragraph.append</code> .	Append content to heading.
<code>clone</code> Use <code>Heading6.clone</code> similar to how you use <code>Paragraph.clone</code> .	Copy heading.

## Examples

### Create Three Levels of Headings

This example shows how to add three levels of headings, each formatted according to its level. This example inserts three heading objects into a document: a `Heading1`, a `Heading2`, and a `Heading3`.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'docx');
open(d);

title = append(d, Paragraph('Document Title'));
title.Bold = true;
title.FontSize = '28pt';

h1 = append(d, Heading1('Chapter 1'));
h1.Style = {PageBreakBefore(true)};
p1 = append(d, Paragraph('Hello World'));

h2 = append(d, Heading2('Section 1.1'));
p2 = append(d, Paragraph('Text for this section. '));

h3 = append(d, Heading3('My Subsection 1.1.a'));
p3 = append(d, Paragraph('Text for this subsection'));

close(d);
rptview(d.OutputPath);
```

### See Also

`mlreportgen.dom.Paragraph` | `mlreportgen.dom.Heading` | `mlreportgen.dom.Heading1` | `mlreportgen.dom.Heading2` | `mlreportgen.dom.Heading3` | `mlreportgen.dom.Heading4` | `mlreportgen.dom.Heading5`

### Topics

“Report Formatting Approaches” on page 13-17

**Introduced in R2016a**

## mlreportgen.dom.Height class

**Package:** mlreportgen.dom

Height of object

### Description

Specifies the height of an image.

### Construction

`heightObj = Height()` creates a format object that specifies a height of 1 inch.

`heightObj = Height(value)` creates a height object having the specified height.

### Input Arguments

#### **value — Height of object**

'lin' (default) | character vector

Height of the object, in the form `valueUnits`, where `Units` is an abbreviation for the units. These abbreviations are valid:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

### Output Arguments

#### **heightObj — Height of object**

mlreportgen.dom.Height object

Height of object, represented by an `mlreportgen.dom.Height` object.

### Properties

#### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

#### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Value — Object height**

lin (default) | character vector

character vector having the format valueUnits, where Units is an abbreviation for the units in which the height is expressed. The following abbreviations are valid:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**See Also**

mlreportgen.dom.RowHeight | mlreportgen.dom.Width

**Topics**

“Report Formatting Approaches” on page 13-17

## mlreportgen.dom.HTML class

**Package:** mlreportgen.dom

Use HTML markup to create DOM document

### Description

Converts a string of HTML markup to DOM objects and appends them to an HTML object that it also creates. You can append the HTML object to a DOM document of any type, including Word and PDF documents.

### Construction

`htmlObj = HTML()` creates an empty HTML object.

`htmlObj = HTML(htmlText)` converts HTML text to an HTML object containing DOM objects having the same content and format.

An HTML object supports these HTML elements and attributes. In addition, HTML objects accept HTML that contains custom CSS properties, which begin with a hyphen. Custom CSS properties are supported in HTML, Microsoft Word, and PDF output.

HTML Element	Attributes
a	class, style, href, name
address	class, style
b	class, style
big	class, style
blockquote	class, style
body	class, style
br	n/a
center	class, style
cite	class, style
code	class, style
dd	class, style
del	class, style
dfn	class, style
div	class, style
dl	class, style
dt	class, style
em	class, style
font	class, style, color, face, size

<b>HTML Element</b>	<b>Attributes</b>
h1, h2, h3, h4, h5, h6	class, style, align
hr	class, style, align
i	class, style
ins	class, style
img	class, style, src, height, width
kbd	class, style
li	class, style
mark	class, style
nobr	class, style
ol	class, style
p	class, style, align
pre	class, style
s	class, style
samp	class, style
small	class, style
span	class, style
strike	class, style
strong	class, style
sub	class, style
sup	class, style
table	class, style, align, bgcolor, border, cellpadding, cellspacing, frame, rules, width
tbody	class, style, align, valign
tfoot	class, style, align, valign
thead	class, style, align, valign
td	class, style, bgcolor, height, width, colspan, rowspan, align, valign, nowrap
th	class, style, bgcolor, height, width, colspan, rowspan, align, valign, nowrap
tr	class, style, align, bgcolor, valign
tt	class, style
u	class, style
ul	class, style
var	class, style

For information about these elements, see <https://developer.mozilla.org/en-US/docs/Web/HTML/Element>.

## Input Arguments

### htmlText — HTML text

character vector

HTML text, specified as a character vector

Example: `html = HTML('<p><b>Hello</b> <i style="color:green"> World</i></p>');`

## Properties

**Note** For HTML markup to display correctly in your report, you must include end tags for empty elements and enclose attribute values in quotation marks. If you want to show a reserved XML markup character as text, you must use its equivalent named or numeric XML character.

Reserved Character	Description	Equivalent Character
>	Greater than	&gt;
<	Less than	&lt;
&	Ampersand	&amp;
"	Double quotation mark	&quot;
'	Single quotation mark	&apos;
%	Percent	&#37;

### Id — ID for HTML object

character vector

A session-unique ID is generated as part of HTML object creation. You can specify an ID to replace the generated ID.

### HTMLTag — Tag name of HTML container element

'div' (default) | character vector

Tag name of HTML container element, specified as a character vector, such as 'div', 'section', or 'article' corresponding to this HTML object. This property applies only to HTML output.

### Children — Children of this HTML object

cell array of `mlreportgen.dom.Element` objects

This read-only property lists child elements that the HTML object contains.

### Parent — Parent of this HTML object

a DOM object

This read-only property lists the parent of this HTML object.

### Style — Formatting to apply to this HTML object

cell array of format objects

Formatting to apply to this HTML object, specified as a cell array of DOM format objects. The children of this HTML object inherit any of these formats that they do not override.



**StyleName — Style name of this HTML object**

character vector

Style name of this HTML object, specified as a character vector. Use a name of a style specified in the style sheet of the document to which this HTML object is appended. The specified style defines the appearance of the HTML object in the output document where not overridden by the formats specified by this StyleName property of the HTML object.

**Tag — Tag for HTML object**

character vector

Tag for HTML object, specified as a character vector.

A session-unique ID is generated as part of HTML object creation. The generated tag has the form CLASS:ID, where CLASS is the class of the element and ID is the value of the Id property of the object. You can specify a tag to replace the generated tag.

Specify your own tag value, for example, to make it easier to identify where an issue occurred during document generation.

**KeepInterElementWhiteSpace — Whether to convert white space between elements**

false (default) | true

Whether to convert white space between elements, specified as true or false. If KeepInterElementWhiteSpace is true, the DOM API converts white space between elements in the input HTML markup to mlreportgen.dom.Text objects. If false, the DOM API ignores white space between elements.

Setting the KeepInterElementWhiteSpace property to true does not preserve white space. To preserve white space, set the KeepInterElementWhiteSpace property to true and add an mlreportgen.dom.WhiteSpace format object to the Style property of the HTML object. The WhiteSpace property of the WhiteSpace object must be set to 'preserve'. For example:

```
import mlreportgen.dom.*
d2 = mlreportgen.dom.Document("ex2", "pdf");
h2 = HTML();
h2.Style = {WhiteSpace('preserve')};
h2.KeepInterElementWhiteSpace = true;
append(h2, '<p>    <span>Hello</span>    <span>World!</span></p>');
append(d2, h2);
close(d2);
rptview(d2)
```

If the input HTML preserves white space using a style attribute, you do not need to use the KeepInterElementWhiteSpace property and WhiteSpace object. For example:

```
import mlreportgen.dom.*
d1 = mlreportgen.dom.Document("ex1", "pdf");
h1 = HTML();
append(h1, '<p style="white-space:pre"> <span>Hello</span>    <span>World!</span></p>');
append(d1, h1);
close(d1);
rptview(d1);
```

If the input HTML uses a CSS style to preserve white space, the DOM API does not preserve the white space unless you use the KeepInterElementWhiteSpace and the WhiteSpace object. For example:

```

import mlreportgen.dom.*
d3 = mlreportgen.dom.Document("ex3", "pdf");
h3 = HTML();
h3.Style = {WhiteSpace('preserve')};
h3.KeepInterElementWhiteSpace = true;
append(h3, [ ...
    '<style type="text/css">.myStyle { white-space: pre}</style>' ...
    '<p class="myStyle"> <span>Hello</span>    <span>World!</span></p>']);
append(d3, h3);
close(d3);
rptview(d3);

```

Alternatively, if the input HTML uses a CSS style to preserve white space, you can prepare the HTML using `mlreportgen.utils.html2dom.prepHTMLString` or `mlreportgen.utils.html2dom.prepHTMLFile`. Then, create an HTML object from the prepared HTML. For example:

```

import mlreportgen.dom.*
d3 = mlreportgen.dom.Document("ex3", "pdf");
h3 = HTML();
htmlStr = ['<style type="text/css">.myStyle { white-space: pre}</style>' ...
    '<p class="myStyle"> <span>Hello</span>    <span>World!</span></p>'];
preppedhtml = mlreportgen.utils.html2dom.prepHTMLString(htmlStr);
append(h3, preppedhtml);
append(d3, h3);
close(d3);
rptview(d3);

```

See “Prepare HTML Before Conversion” on page 13-105.

### EMBaseFontSize — Font size of one em unit in points

12 (default) | integer

Font size of one em unit, in points, specified as an integer. If a style in the HTML text specifies font size in em units, the number of em units is multiplied by the value of the `EMBaseFontSize` property to determine the font size in points. For example, the following code results in a font size of 20 points.

```

h = HTML();
h.EMBaseFontSize = 10;
append(h, '<p style="font-size:2em">Hello</p>');

```

Set the `EMBaseFontSize` property in an empty `mlreportgen.dom.HTML` object. Then add the HTML to the object. For example:

```

import mlreportgen.dom.*;
rpt = Document('MyReport', 'pdf');
htmlobj = HTML();
htmlobj.EMBaseFontSize = 14;
append(htmlobj, '<p style="font-size:2em">Hello</p>');
append(rpt, htmlobj);
close(rpt);
rptview('MyReport.pdf');

```

Setting the `EMBaseFontSize` property in an `mlreportgen.dom.HTML` object that already contains the HTML has no effect.

## Methods

append	Append HTML string to HTML object
clone	Copy HTML object

## Examples

### Convert HTML Markup to a Word Report

Create an `mlreportgen.dom.HTML` object from HTML markup and add it to a Word report.

```
import mlreportgen.dom.*;
rpt = Document('MyReport', 'docx');
html = HTML('<p><b>Hello</b> <i style="color:green"> World</i></p>');
append(html, '<p>This is <u>me</u> speaking</p>');
append(rpt, html);
close(rpt);
rptview(rpt.OutputPath);
```

The resulting Word report looks like this:

Hello *World*

This is me speaking

## Tips

- MATLAB Report Generator `mlreportgen.dom.HTML` or `mlreportgen.dom.HTMLFile` objects typically cannot accept the raw HTML output of third-party applications, such as Microsoft Word, that export native documents as HTML markup. In these cases, your Report API report generation program can use the `mlreportgen.utils.html2dom.prepHTMLString` and `mlreportgen.utils.html2dom.prepHTMLFile` functions to prepare the raw HTML for use with the `mlreportgen.dom.HTML` or `mlreportgen.dom.HTMLFile` objects. Typically, your program will have to further process the prepared HTML to remove valid but undesirable objects, such as line feeds that were in the raw content.
- Word and PDF documents require inline elements, such as text and links, to be contained in a paragraph. To meet this requirement, the HTML parser creates wrapper paragraphs to contain inline elements that are not already in a paragraph. If you create an `mlreportgen.dom.HTML` or `mlreportgen.dom.HTMLFile` object from HTML that contains inline elements that are not in paragraphs and add the object to an HTML document, the generated HTML can differ from the input HTML. To generate the inline elements without the added wrapper paragraphs, insert the HTML markup into an HTML document by using an `mlreportgen.dom.RawText` object.
- By default, the DOM API uses a base font size of 12 points to convert `em` units to actual font sizes. For example, a font size specified as `2em` converts to 24 points. To specify a different base font size, add your content to a report by using an `mlreportgen.dom.HTML` object. Set the `EMBaseFontSize` property of the object to the base font size. For example, if you set the `EMBaseFontSize` property to 14, a font size of `2em` converts to 28 points.

### See Also

`mlreportgen.dom.HTMLFile` | `addHTML` | `mlreportgen.utils.html2dom.prepHTMLFile` | `mlreportgen.utils.html2dom.prepHTMLString`

### Topics

“Convert HTML Content to a DOM Object” on page 13-105

“Prepare HTML for Conversion to a DOM Object” on page 13-108

“Requirements for Converting HTML to a DOM Object” on page 13-110

### External Websites

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element>

<https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>

### Introduced in R2015a

# mlreportgen.dom.HTMLFile class

**Package:** mlreportgen.dom

Convert an HTML file to a DOM document

## Description

Converts the contents of an HTML file to an `mlreportgen.dom.HTMLFile` object containing DOM objects having the same content and format. You can append the `HTMLFile` object to a DOM document of any type, including Word and PDF documents.

## Construction

`htmlFileObj = HTMLFile(htmlFile)` converts the HTML file to an `HTMLFile` object containing DOM objects having the same content and format.

An `HTMLFile` object supports these HTML elements and attributes. In addition, `HTMLFile` objects accept HTML that contains custom CSS properties, which begin with a hyphen. Custom CSS properties are supported in HTML, Microsoft Word, and PDF output.

HTML Element	Attributes
a	class, style, href, name
address	class, style
b	class, style
big	class, style
blockquote	class, style
body	class, style
br	n/a
center	class, style
cite	class, style
code	class, style
dd	class, style
del	class, style
dfn	class, style
div	class, style
dl	class, style
dt	class, style
em	class, style
font	class, style, color, face, size
h1, h2, h3, h4, h5, h6	class, style, align

HTML Element	Attributes
hr	class, style, align
i	class, style
ins	class, style
img	class, style, src, height, width
kbd	class, style
li	class, style
mark	class, style
nobr	class, style
ol	class, style
p	class, style, align
pre	class, style
s	class, style
samp	class, style
small	class, style
span	class, style
strike	class, style
strong	class, style
sub	class, style
sup	class, style
table	class, style, align, bgcolor, border, cellpadding, cellspacing, frame, rules, width
tbody	class, style, align, valign
tfoot	class, style, align, valign
thead	class, style, align, valign
td	class, style, bgcolor, height, width, colspan, rowspan, align, valign, nowrap
th	class, style, bgcolor, height, width, colspan, rowspan, align, valign, nowrap
tr	class, style, align, bgcolor, valign
tt	class, style
u	class, style
ul	class, style
var	class, style

For information about these elements, see <https://developer.mozilla.org/en-US/docs/Web/HTML/Element>.

These CSS formats are supported:

- background-color

- border
- border-bottom
- border-bottom-color
- border-bottom-style
- border-bottom-width
- border-color
- border-left
- border-left-color
- border-left-style
- border-left-width
- border-right
- border-right-color
- border-right-style
- border-right-width
- border-style
- border-top
- border-top-color
- border-top-style
- border-top-width
- border-width
- color
- counter-increment
- counter-reset
- display
- font-family
- font-size
- font-style
- font-weight
- height
- line-height
- list-style-type
- margin
- margin-bottom
- margin-left
- margin-right
- margin-top
- padding
- padding-bottom
- padding-left

- padding-right
- padding-top
- text-align
- text-decoration
- text-indent
- vertical-align
- white-space
- width

For information about these formats, <https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>.

### Input Arguments

#### htmlFile — HTML file path

character vector

HTML file path, specified as a character vector.

### Properties

**Note** For HTML markup to display correctly in your report, you must include end tags for empty elements and enclose attribute values in quotation marks. If you want to show a reserved XML markup character as text, you must use its equivalent named or numeric XML character.

Reserved Character	Description	Equivalent Character
>	Greater than	&gt;
<	Less than	&lt;
&	Ampersand	&amp;
"	Double quotation mark	&quot;
'	Single quotation mark	&apos;
%	Percent	&#37;

#### Id — ID for HTMLFile object

character vector

A session-unique ID is generated as part of HTMLFile object creation. You can specify an ID to replace the generated ID.

#### HTMLTag — HTML tag name of HTML container element

'div' (default) | character vector

Tag name of HTML container element, specified as a character vector, such as 'div', 'section', or 'article' corresponding to this HTMLFile object. This property applies only to HTML output.

#### Children — Children of this HTMLFile object

cell array of mlreportgen.dom.Element objects



This read-only property lists child elements that the HTMLFile object contains.

### Parent — Parent of this HTMLFile object

a DOM object

This read-only property lists the parent of this HTMLFile object.

### Style — Formatting to apply to HTMLFile object

cell array of format objects

Formatting to apply to the HTMLFile object, specified as a cell array of DOM format objects. The children of this HTMLFile object inherit any of these formats that they do not override.

### StyleName — Style name of HTMLFile object

character vector

Style name of this HTMLFile object, specified as a character vector. Use a name of a style specified in the style sheet of the document to which this HTMLFile object is appended. The specified style defines the appearance of the HTMLFile object in the output document where not overridden by the formats specified by this StyleName property of the HTMLFile object.

### Tag — Tag for HTMLFile object

character vector

Tag for HTMLFile object, specified as a character vector.

A session-unique ID is generated as part of HTMLFile object creation. The generated tag has the form CLASS:ID, where CLASS is the class of the element and ID is the value of the Id property of the object. You can specify a tag to replace the generated tag.

Specify your own tag value, for example, to make it easier to identify where an issue occurred during document generation.

---

**Note** HTMLFile ignores the KeepInterElementWhiteSpace property. If you want to preserve white space, use fileread to read your HTML file as text and then follow the procedure described for the mlreportgen.dom.HTMLKeepInterElementWhiteSpace property.

---

## Methods

append                      Append HTML to HTMLFile object

## Examples

### Convert HTML File to a Word Report

Create a text file named myHTML.html and save it in the current folder. Add this text into the file:

```
<html>
<head>
<style>p {font-size:14pt;}</style>
</head>
<body>
<p style='white-space:pre'><b>Hello</b><i style='color:green'> World</i></p>
<p>This is <u>me</u> speaking</p>
</body>
</html>
```

To convert the `myHTML.html` file to a Word report, run these commands:

```
import mlreportgen.dom.*;
rpt = Document('MyReport','docx');
htmlFile = HTMLFile('myHTML.html');
append(rpt,htmlFile);
close(rpt);
rptview(rpt.OutputPath);
```

The resulting Word report contains the text that you specified in the HTML file.

Hello *World*¶

This is me speaking¶

## Tips

- MATLAB Report Generator `mlreportgen.dom.HTML` or `mlreportgen.dom.HTMLFile` objects typically cannot accept the raw HTML output of third-party applications, such as Microsoft Word, that export native documents as HTML markup. In these cases, your Report API report generation program can use the `mlreportgen.utils.html2dom.prepHTMLString` and `mlreportgen.utils.html2dom.prepHTMLFile` functions to prepare the raw HTML for use with the `mlreportgen.dom.HTML` or `mlreportgen.dom.HTMLFile` objects. Typically, your program will have to further process the prepared HTML to remove valid but undesirable objects, such as line feeds that were in the raw content.
- Word and PDF documents require inline elements, such as text and links, to be contained in a paragraph. To meet this requirement, the HTML parser creates wrapper paragraphs to contain inline elements that are not already in a paragraph. If you create an `mlreportgen.dom.HTML` or `mlreportgen.dom.HTMLFile` object from HTML that contains inline elements that are not in paragraphs and add the object to an HTML document, the generated HTML can differ from the input HTML. To generate the inline elements without the added wrapper paragraphs, insert the HTML markup into an HTML document by using an `mlreportgen.dom.RawText` object.
- By default, the DOM API uses a base font size of 12 points to convert `em` units to actual font sizes. For example, a font size specified as `2em` converts to 24 points. To specify a different base font size, add your content to a report by using an `mlreportgen.dom.HTML` object. Set the `EMBaseFontSize` property of the object to the base font size. For example, if you set the `EMBaseFontSize` property to 14, a font size of `2em` converts to 28 points.

## See Also

`mlreportgen.dom.HTML` | `addHTMLFile` | `mlreportgen.utils.html2dom.prepHTMLFile` | `mlreportgen.utils.html2dom.prepHTMLString`

## Topics

“Convert HTML Content to a DOM Object” on page 13-105

“Prepare HTML for Conversion to a DOM Object” on page 13-108

“Requirements for Converting HTML to a DOM Object” on page 13-110

## External Websites

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element>

<https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>

**Introduced in R2015a**

## mlreportgen.dom.HorizontalRule class

**Package:** mlreportgen.dom

Horizontal line between report content

### Description

Horizontal line to visually separate report content in a report. You can append a `HorizontalRule` object to these objects:

- `mlreportgen.dom.Document`
- `mlreportgen.dom.DocumentPart`
- `mlreportgen.dom.TableEntry`
- `mlreportgen.dom.Group`
- `mlreportgen.dom.Container`

### Construction

`horizontalRuleObj = HorizontalRule()` creates an unspecified horizontal line.

### Output Arguments

**horizontalRuleObj — Horizontal line**

`mlreportgen.dom.HorizontalRule` object

Horizontal line, returned as an `mlreportgen.dom.HorizontalRule` object.

### Properties

**Border — Line style for horizontal rule**

character vector

Line style for horizontal rule, specified as one of these values.

Value	Applies To	
	DOCX	HTML and PDF
'dashed'	✓	✓
'dashdotstroked'	✓	
'dashsmallgap'	✓	
'dotted'	✓	✓
'dotdash'	✓	

Value	Applies To	
	DOCX	HTML and PDF
'dotdotdash'	✓	
'double'	✓	✓
'doublewave'	✓	
'inset'	✓	✓
'none'	✓	✓
'outset'	✓	✓
'single'	✓	
'solid'		✓
'thick'	✓	
'thickthinlargegap'	✓	
'thickthinmediumgap'	✓	
'thickthinsmallgap'	✓	
'thinthicklargegap'	✓	
'thinthickmediumgap'	✓	
'thinthicksmallgap'	✓	
'thinthickthinlargegap'	✓	
'thinthickthinmediumgap'	✓	
'thinthickthinsmallgap'	✓	
'threedemboss'	✓	
'threedengrave'	✓	
'triple'	✓	
'wave'	✓	

**BorderColor — Color of line**

character vector

Color of the line, specified as a character vector. You can specify:

- The name of a color. The name must be a CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- A hexadecimal RGB (truecolor) value as #RRGGBB. For example, #0000ff is a shade of blue.

**BorderWidth — Width of line (in HTML report)**

character vector

Width of line (in an HTML report), specified in the format `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**BackgroundColor — Background color of line (in HTML report)**

character vector

Background color of the line, specified as a character vector. You can specify:

- The name of a color. The name must be a CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- A hexadecimal RGB (truecolor) value as #RRGGBB. For example, #0000ff is a shade of blue.

This property applies only to HTML reports.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Style — Format specification for line**

array of format objects

Format specification for the line, specified as an array of format objects.

**StyleName — Style sheet style for line**

character vector

Style sheet style for line, specified as a character vector. The name of a style must be specified in the style sheet of the document or document part to which this element is appended. The specified style defines the appearance of this element in the output document where not overridden by the formats specified by the `Style` property of this element.

**Tag — Tag for line**

character vector

Tag for line, specified as a character vector.

A session-unique ID is generated as part of document element creation. The generated tag has the form CLASS:ID, where CLASS is the class of the element and ID is the value of the Id property of the object. You can specify a tag to replace the generated tag.

To make it easier to identify where an issue occurred during document generation, you can Specify your own tag value.

## Methods

Method	Purpose
clone	Copy horizontal line.
Use HorizontalRule.clone in a similar way to how you use Paragraph.clone.	

## Examples

### Add a Horizontal Rule

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('horizontalRule',doctype);
p1 = Paragraph('Top content');
append(d,p1);

hr = HorizontalRule();
hr.Border = 'dotted';
hr.BorderColor = 'blue';
append(d,hr);

p2 = Paragraph('Bottom content');
append(d,p2);

close(d);
rptview(d.OutputPath);
```

## Compatibility Considerations

### Background Color Not Supported for PDF Reports

*Behavior changed in R2021a*

Starting in R2021a, you cannot specify the background color of a horizontal rule in a PDF report.

### Introduced in R2015b

## mlreportgen.dom.Hyphenation class

**Package:** mlreportgen.dom

Enable hyphenation

### Description

Use an object of the `mlreportgen.dom.Hyphenation` class to specify the hyphenation behavior for paragraphs and table cells. A `Hyphenation` object affects only a PDF report.

By default, hyphenation is disabled for paragraphs and table cells. When hyphenation is disabled, where possible, a line break occurs between words. When necessary to prevent overflow, a line break may occur anywhere in a word.

When hyphenation is enabled, a line break occurs at the end of a syllable and a hyphenation character is inserted. If a table cell contains a long sequence of numbers or letters that have no clear syllable breaks, overflow can occur. The table stretches to accommodate the overflow.

The `mlreportgen.dom.Hyphenation` class is a `handle` class.

### Class Attributes

<code>ConstructOnLoad</code>	<code>true</code>
<code>HandleCompatible</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`h = mlreportgen.dom.Hyphenation()` enables hyphenation and sets the `Value` property to `' - '`.

`h = mlreportgen.dom.Hyphenation(type)` specifies whether to hyphenate and the hyphenation character to use.

### Input Arguments

#### **type** — Type of hyphenation

`false | true | ' - ' | ' '`

Type of hyphenation, specified as one of these values:

- `true` — Enables hyphenation and sets the `Value` property to `' - '`.
- `' - '` — Enables hyphenation and sets the `Value` property to `' - '`.
- `' '` — Enables hyphenation and sets the `Value` property to a space (`' '`).
- `false` — Disables hyphenation and sets the `Value` property to `[]`.



## Properties

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Value — Hyphenation character

'-' (default) | true | ' ' | false | []

Hyphenation character, specified as one of these values:

- `true` — Enables hyphenation and uses a hyphen (-) for the hyphenation character.
- `' - '` — Enables hyphenation and uses a hyphen (-) for the hyphenation character.
- `' '` — Enables hyphenation and uses a space for the hyphenation character.
- `false` — Disables hyphenation.
- `[]` — Disables hyphenation.

## Examples

### Enable Hyphenation in a PDF Table

To enable hyphenation in a PDF table, add a `Hyphenation` object to the `TableEntriesStyle` property of the `mlreportgen.dom.Table` object. Use the default hyphenation character, `' - '`.

```
import mlreportgen.dom.*
d = Document('myreport', 'pdf');
open(d);
data = '/mylongpath/hyphenation/example/myveryveryveryverylongpathname.doc';
table = Table({data});
table.Width = '2in';
table.TableEntriesStyle = {Hyphenation(true)};
append(d, table);
close(d);
rptview(d);
```

## Compatibility Considerations

### Behavior change for hyphenation

*Behavior changed in R2020b*

Starting in R2020b:

- Hyphenation is disabled, by default, for paragraphs and tables.
- When hyphenation is disabled, where possible, a line break occurs between words. If necessary to prevent overflow, a line break can occur anywhere in a word.
- When hyphenation is enabled, a line break and hyphenation character occur only at the end of a syllable. If a table cell contains a long sequence of numbers or letters that have no clear syllable breaks, overflow can occur. The table stretches to accommodate the overflow.

Before R2020b:

- Hyphenation was disabled for paragraphs. Hyphenation was enabled for tables and the default hyphenation character was an empty space.
- When hyphenation was disabled, a line break occurred only between words, which resulted in an overflow when a word extended past the boundary of a page or table cell.
- When hyphenation was enabled, a line break and hyphenation character could occur anywhere in a word in a paragraph or table.

## **See Also**

`mlreportgen.dom.PDFPageLayout` | `mlreportgen.dom.TableEntry`

## **Topics**

“Hyphenation Styles in PDF Templates” on page 13-141

## **Introduced in R2016b**

# mlreportgen.dom.Image class

**Package:** mlreportgen.dom mlreportgen.dom mlreportgen.dom

Image to include in report

## Description

Use an object of the `mlreportgen.dom.Image` class to include an image in a report.

The `mlreportgen.dom.Image` class is a `handle` class.

## Class Attributes

<code>ConstructOnLoad</code>	<code>true</code>
<code>HandleCompatible</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`imageObj = mlreportgen.dom.Image(imagePath)` creates an `mlreportgen.dom.Image` object with the `Path` property set to the path of an image.

---

**Note** The contents of the specified image file are copied into the output document when the document is closed. Do not delete or overwrite the image file before closing the document. If you create an image file and the corresponding `mlreportgen.dom.Image` object in a loop, for each loop iteration, use a unique file name for the image file.

---

## Input Arguments

### **imagePath** — Path of image file

character vector | string scalar

Path of an image file to include in a report, specified as a character vector or string scalar. You can use the image formats in the table.

Image Format	File Extension	Supported Document Type		
		HTML	Word	PDF
Windows bitmap	.bmp	✓	✓	✓
Windows metafile	.emf		✓	

Image Format	File Extension	Supported Document Type		
		HTML	Word	PDF
Graphics Interchange Format	.gif	✓	✓	✓
JPEG image	.jpg	✓	✓	✓
PDF	.pdf			✓
Portable Network Graphics	.png	✓	✓	✓
Scalable Vector Graphics	.svg	✓	✓	✓
TIFF image	.tif		✓	✓

## Properties

### Path — Path of image file

character vector

Path of the image file, specified as a character vector. This property is read-only.

### Height — Image height

character vector | string scalar

Image height, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, "2in" specifies two inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points
- % — percent

When the Image object is created, the Height property is set to a value based on the image file specified by the Path property.

### Width — Image width

character vector | string scalar

Image width, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, "2in" specifies two inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points
- % — percent

When the Image object is created, the Width property is set to a value based on the image file specified by the Path property.

### **Map — Map of hyperlink areas in image**

mlreportgen.dom.ImageMap object

Map of hyperlink areas in this image, specified as an mlreportgen.dom.ImageMap object. The Map property applies to only HTML and PDF reports.

See “Create Image Maps” on page 13-99.

### **StyleName — Name of image style**

character vector | string scalar

Name of image style, specified as a character vector or string scalar. The StyleName value must be a style in the style sheet of the document or document part to which this image is appended. Format objects in the Style property of this Image object override the style in the style sheet.

---

**Note** The StyleName property is ignored for Word output.

---

### **Style — Formats that define image style**

cell array of DOM format objects

Formats that define the style of this image, specified as a cell array of DOM format objects. The formats override the corresponding formats defined by the style specified by the StyleName property. See “Create and Format Images” on page 13-88.

### **CustomAttributes — Custom attributes of document element**

array of mlreportgen.dom.CustomAttribute objects

Custom attributes of this document element, specified as an array of mlreportgen.dom.CustomAttribute objects. The custom attributes must be supported by the output format.

### **Parent — Parent of image**

document element object

Parent of this image, specified as a document element object. This property is read-only.

### **Children — This property is ignored**

array of document element objects

This property is ignored.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Methods****Public Methods**

Method	Purpose
append	Do not use this method. Appending content to an <code>mlreportgen.dom.Image</code> object has no effect in the generated output.
clone	<code>clone = clone(imageObj)</code> creates a copy of the specified <code>mlreportgen.dom.Image</code> object.  To append the same image to different parts of a document, use the <code>clone</code> method. Do not add the same <code>mlreportgen.dom.Image</code> object to a document more than one time.

**Examples****Include Image in a Report**

Use an `mlreportgen.dom.Image` object to specify an image that you want to include in a report. Include the image in the report by appending the `Image` object to the report.

Import the DOM package so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
```

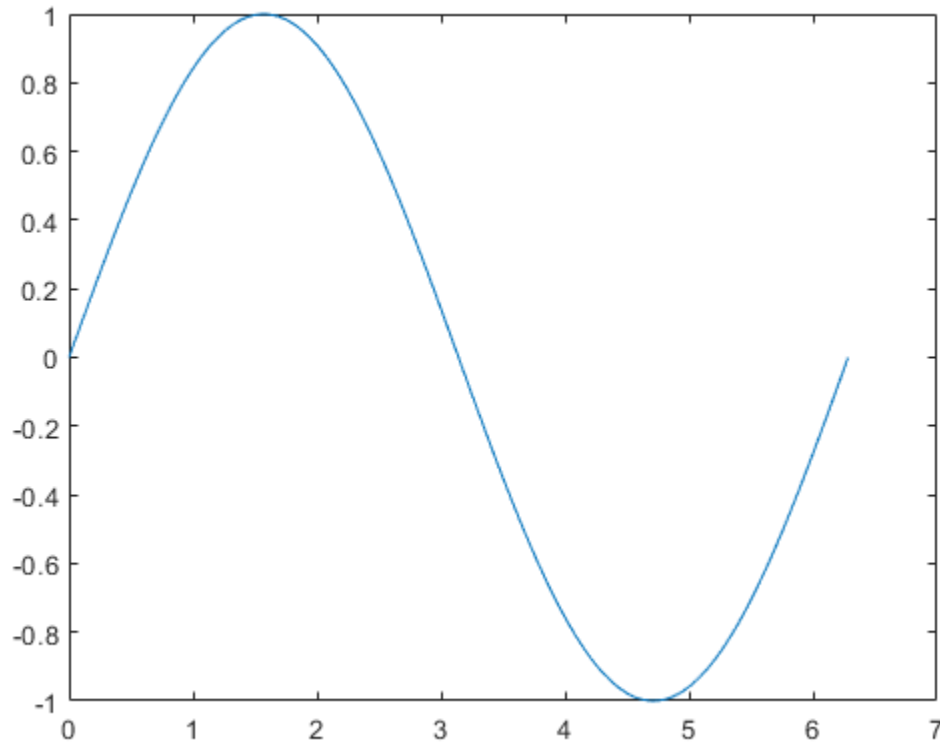
Create a report and add a title for the image using an `mlreportgen.dom.Paragraph` object.

```
d = Document("myImageReport", "docx");
```

```
p = Paragraph("Plot 1");
p.Bold = true;
append(d,p);
```

Save a plot as an image file.

```
x = 0:pi/100:2*pi;  
y = sin(x);  
plot(x,y);  
  
saveas(gcf,"myPlot_img.png");
```



Create an `mlreportgen.dom.Image` object that specifies the path of the image file. Specify that the width and height are 4 inches. Append the Image object to the report.

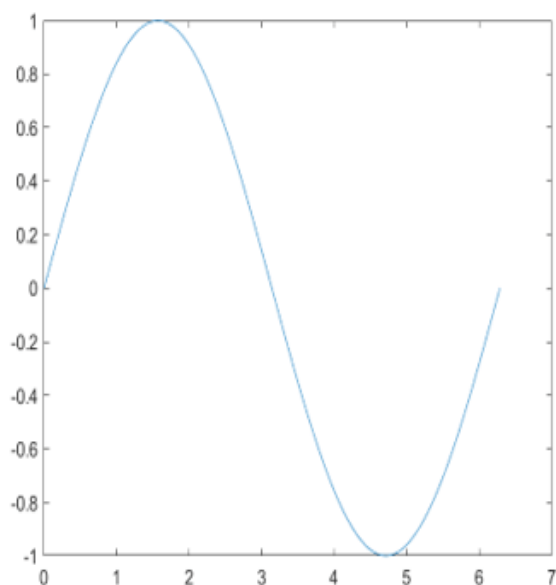
```
plot1 = Image("myPlot_img.png");  
plot1.Width = "4in";  
plot1.Height = "4in";  
append(d,plot1);
```

Close and view the report.

```
close(d);  
rptview(d);
```

Here is the image in the generated report:

Plot 1



## Compatibility Considerations

### SVG image support for Word reports

*Behavior changed in R2019b*

Starting in R2019b, you can include Scalable Vector Graphics (SVG) images in Word reports. Word reports that contain SVG images require Microsoft Word 2016 or a later version. To generate reports with images that are compatible with earlier versions of Word, create `mlreportgen.dom.Image` objects from images that have a format other than SVG.

### See Also

`mlreportgen.dom.ScaleToFit` | `mlreportgen.dom.ImageMap` |  
`mlreportgen.dom.ImageArea` | `mlreportgen.dom.Width` | `mlreportgen.dom.Height`

### Topics

“Create and Format Images” on page 13-88  
“Create Image Maps” on page 13-99  
“Scale Image to Fit Page” on page 17-24  
“Side-by-Side Images” on page 17-20  
“Center Image on a Page” on page 17-121  
“Hyperlink Image” on page 17-26



**Introduced in R2014b**

## mlreportgen.dom.ImageArea class

**Package:** mlreportgen.dom

Define image area as hyperlink

### Description

Define an area in an image to hyperlink to. When you click an image area, an HTML browser displays the target page, based on the URL or link target you specify. If the target is in the same document, clicking the link moves you to that location in the document. You can provide alternative text for screen readers that support alternative text. Create image areas in reports with PDF or HTML output. Create the image map using `mlreportgen.dom.ImageMap` and append areas to the map.

### Construction

`imageAreaObj = ImageArea()` creates an empty image area.

`imageAreaObj = ImageArea(target, altText, x1, y1, x2, y2)` creates a rectangular image area.

`imageAreaObj = ImageArea(target, altText, x, y, radius)` creates a circular image area.

`imageAreaObj = ImageArea(target, altText, polygonCoordinates)` creates a polygonal image area.

### Input Arguments

#### **target — Image area hyperlink target**

character vector

Image area hyperlink target, specified as either:

- URL of the page to be loaded when this image area is clicked
- Name of a link target

#### **altText — Text to display if image is not visible**

character vector

Text to display if the image is not visible, specified as a character vector.

#### **x1 — x coordinate of top-left corner of rectangular image area, in pixels**

unsigned integer

Specify relative to the top-left corner of the image.

Data Types: `uint16`

#### **y1 — y coordinate of top-left corner of rectangular image area**

unsigned integer

Specify relative to the top-left corner of the image, in pixels.

Data Types: uint16

**x2 — x coordinate of bottom-right corner of rectangular image area**

unsigned integer

Specify relative to the top-left corner of the image, in pixels.

Data Types: uint16

**y2 — y coordinate of bottom-right corner of rectangular image area**

unsigned integer

Specify relative to the top-left corner of the image, in pixels.

Data Types: uint16

**x — x coordinate of center of circular image area**

unsigned integer

Specify relative to the top-left corner of the image, in pixels.

Data Types: uint16

**y — y coordinate of center of circular image area**

unsigned integer

Specify relative to the top-left corner of the image, in pixels.

Data Types: uint16

**radius — Radius of circular image area**

unsigned integer

The radius, in pixels.

Data Types: uint16

**polygonCoordinates — Coordinates of polygonal image area**

array

Specify an array of x and y coordinate pairs, with coordinates for each corner of the polygon, in the form [x1, y1, x2, y2, ... xN, yN]. Specify the coordinates to reflect the corners of the polygon, in sequence.

Specify each coordinate relative to the top-left corner of the image, in pixels.

**Output Arguments**

**imageAreaObj — Image area hyperlink**

mlreportgen.dom.ImageArea object

Image area hyperlink, represented by an mlreportgen.dom.ImageArea object.

**Properties**

**Target — Image area target**

character vector

Image area target, specified as either:

- URL of the page to be loaded when this image area is clicked
- Name of a link target

**AlternateText — Text to display if image is not visible**

character vector

Text to display if the image is not visible, specified as a character vector.

**Shape — Shape of image area**

'rect' | 'circle' | 'poly'

(Read-only) Possible values are:

- 'rect' — rectangular image area
- 'circle' — circular image area
- 'poly' — polygonal image area

**Coords — Coordinates for image area**

array

(Read-only) The coordinates represent different kinds of points, depending on the shape of the image area. Coordinates are relative to the top-left corner of the image.

- For a rectangle, the coordinates represent the top-left corner and the bottom-right corner.
- For a circle, the array represents the coordinates at the center of the circle and the radius.
- For a polygon, the coordinates represent the corners.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Examples

**Add Image Area to Image of a MATLAB Plot**

```
import mlreportgen.dom.*  
d = Document('imageArea', 'pdf');
```

```
% Create a plot and save it as an image file
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y);
annotation('textbox', [0.2,0.4,0.1,0.1],...
    'string', 'Help on plot function');
saveas(gcf,'plot_img.png');

% Create the DOM image object and append it to your document
plot1 = Image('plot_img.png');
append(d,plot1);

% Define the area and link target using ImageArea
target = ['https://www.mathworks.com/help/matlab/ref/' ...
    'plot.html?searchHighlight=plot'];
areal = ImageArea( target, ...
    'plot function help',160,340,383,392);

% Create the image map object and append the area to it
map = ImageMap();
append(map,areal);
plot1.Map = map;

close(d);
rptview(d.OutputPath);
```

## See Also

[mlreportgen.dom.ImageMap](#) | [mlreportgen.dom.Image](#) | [mlreportgen.dom.LinkTarget](#)

## Topics

“Create and Format Images” on page 13-88

## mlreportgen.dom.ImageMap class

**Package:** mlreportgen.dom

Map of hyperlink areas in image

### Description

Map of image areas, which are areas in an image that you can click to open content in a browser or to navigate to another location in the same page. You can create image maps in reports with PDF or HTML output. Define areas using `mlreportgen.dom.ImageArea` and append them to the map.

### Construction

`map = ImageMap()` creates an empty image map. Use the `ImageMap.append` method to add image areas to the map.

### Output Arguments

**map** — Map of hyperlink areas in image

`mlreportgen.dom.ImageMap` object

Map of hyperlink areas in image, returned as an `mlreportgen.dom.ImageMap` object.

### Properties

**Id** — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Tag** — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Methods

Method	Purpose
<code>append</code>	Append an image area to this image map.

Method	Purpose
clone	Clone this image map.
Use ImageMap.clone in a similar way you how you use Paragraph.clone.	

## Examples

### Append an Image Area to an Image Map

Define an ImageArea object that specifies the size and location of the area and the action that occurs when you click the area. Then append the area to an ImageMap object.

```
import mlreportgen.dom.*
d = Document('imageArea', 'pdf');

% Create a plot and save it as an image file
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y);
annotation('textbox', [0.2,0.4,0.1,0.1],...
           'string', 'Help on plot function');
saveas(gcf, 'plot_img.png');

% Create the DOM image object and append it to your document
plot1 = Image('plot_img.png');
append(d, plot1);

% Define the area and link target using ImageArea
target = ['https://www.mathworks.com/help/matlab/ref/' ...
         'plot.html?searchHighlight=plot'];
areal = ImageArea( target, ...
                  'plot function help', 160, 340, 383, 392);

% Create the image map object and append the area to it
map = ImageMap();
append(map, areal);
plot1.Map = map;

close(d);
rptview(d.OutputPath);
```

### See Also

mlreportgen.dom.ImageArea | mlreportgen.dom.Image

### Topics

“Create and Format Images” on page 13-88

## mlreportgen.dom.InnerMargin class

**Package:** mlreportgen.dom

Margin between content and bounding box

### Description

Specifies the margin between the content and the bounding box of a document object. A bounding box of an object includes the border of the object (if it has a border), the inner margin, and the object content.

### Construction

`marginObj = InnerMargin()` creates an unspecified margin between the content of an object and its bounding box.

`marginObj = InnerMargin(all)` creates the specified margin on all sides between the content of an object and its bounding box.

`marginObj = InnerMargin(left, right)` creates the specified margins between the left and right sides of the content of an object and its bounding box.

`marginObj = InnerMargin(left, right, top, bottom)` creates the specified margins between sides of the content of an object and its bounding box.

### Input Arguments

#### **all** — Margin size on all sides

character vector

Margin on all sides between the content of an object and its bounding box in the form `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

#### **left** — Left margin size

character vector

Left margin between the content of an object and its bounding box. See the `all` input argument description for valid values.

#### **right** — Right margin size

character vector



Right margin between the content of an object and its bounding box. See the `all` input argument description for valid values.

**top — Top margin size**

character vector

Top margin between the content of an object and its bounding box. See the `all` input argument description for valid values.

**bottom — Bottom margin size**

character vector

Bottom margin between the content of an object and its bounding box. See the `all` input argument description for valid values.

**Output Arguments****marginObj — Margin between content and bounding box**

mlreportgen.dom.InnerMargin object

Margin between content and bounding box, specified with an `mlreportgen.dom.InnerMargin` object.

**Properties****Bottom — Size of bottom margin**

character vector

Size of bottom margin in the form `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**Left — Size of left margin**

character vector

Left margin between the content of an object and its bounding box. See the `Bottom` property description for valid value

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Right — Size of right margin**

character vector

Right margin between the content of an object and its bounding box. See the `Bottom` property description for valid values

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Top — Size of top margin

character vector

Top margin between the content of an object and its bounding box. See the `Bottom` property description for valid values.

## Examples

### Add Inner Margins to a Paragraph

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);

p = Paragraph('Hello World');
p.Style = {Border('solid','red'), ...
    HAlign('center'),InnerMargin('12pt')};
append(d,p);

p = Paragraph('More Greetings');
p.Style = {Border('solid','blue'), ...
    HAlign('center'),InnerMargin('30pt')};

append(d,p);
close(d);
rptview('test',doctype);
```

### See Also

`mlreportgen.dom.OuterMargin`

### Topics

“Report Formatting Approaches” on page 13-17

# mlreportgen.dom.InternalLink class

**Package:** mlreportgen.dom

Hyperlink to a location in same document

## Description

Hyperlink to a location in the same document that contains the hyperlink. Use this kind of link to provide internal navigation within a document.

## Construction

`internalLinkObj = InternalLink(targetName,linkText)` creates a hyperlink to the specified link target object and uses the specified link text.

`internalLinkObj = InternalLink(targetName,linkText,linkTextStyleName)` creates a hyperlink to the specified link target and uses the specified style name for the link text.

`internalLinkObj = InternalLink(targetName,textObj)` creates a hyperlink to the specified target using the specified Text object.

## Input Arguments

### **targetName — Link target name**

character vector

Link target name, specified as character vector. The character vector is the value in the Name property of an mlreportgen.dom.LinkTarget object or a URL.

### **linkText — Link text**

character vector

The text to use for the link text.

### **linkTextStyleName — Name of style for link text**

character vector

Name of style to use for the link text.

### **textObj — Text object containing link text**

mlreportgen.dom.Text object

Text object containing link text, specified by an mlreportgen.dom.Text object.

## Output Arguments

### **internalLinkObj — Internal link**

mlreportgen.dom.InternalLink object

Internal link, represented by an mlreportgen.dom.InternalLink object.

## Properties

### **Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

### **CustomAttributes — Custom attributes of document element**

array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this document element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. The custom attributes must be supported by the output format.

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

### **StyleName — Name of link style defined in the template**

character vector

Name of link style defined in the template, specified as a character vector. The style specified by `styleName` must be defined in the template used to create the document to which the link is appended.

### **Style — Format specification**

array of format objects

Format objects that specify the format of a document element.

### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### **Target — Internal target link**

character vector

This read-only property displays the link target of this hyperlink.

## Methods

Method	Purpose
<b>append</b> Use <code>InternalLink.append</code> in a similar way to how you use <code>ExternalLink.append</code> .	Append text or a <code>Text</code> , <code>Image</code> , or <code>CustomElement</code> object.
<b>clone</b> Use <code>InternalLink.clone</code> in a similar way to how you use <code>Paragraph.clone</code> .	Copy the internal link.

## Examples

### Add Internal Link

Link to a target in the same document by using an `mlreportgen.dom.LinkTarget` object to define the link target and an `mlreportgen.dom.InternalLink` object to define the link. The link target name associates the link with the link target. To generate a link target name that is valid for all report types, use `mlreportgen.utils.normalizeLinkID`.

```
import mlreportgen.dom.*
import mlreportgen.utils.*
d = Document('mydoc', 'pdf');

% Append a link target to a heading
h = Heading(1, 'Author's Biography');
h.Style = {PageBreakBefore(true)};
linkID = normalizeLinkID('bio');
append(h, LinkTarget(linkID));

% Link to the target
append(d, InternalLink(linkID, 'About the Author'));

% Append the heading
append(d, h);

close(d);
rptview(d);
```

### See Also

`mlreportgen.dom.LinkTarget` | `mlreportgen.dom.ExternalLink` |  
`mlreportgen.utils.normalizeLinkID`

### Topics

“Create Links” on page 13-82

## mlreportgen.dom.Italic class

**Package:** mlreportgen.dom

Italic for mlreportgen.dom.Text object

### Description

Use an object of the mlreportgen.dom.Italic class to italicise the text in a Text object.

The mlreportgen.dom.Italic class is a handle class.

### Class Attributes

HandleCompatible	true
ConstructOnLoad	true

## Creation

### Description

`italicObj = mlreportgen.dom.Italic()` creates a format object to italicize the text in a Text object.

`italicObj = mlreportgen.dom.Italic(value)` sets the Value property to `value`. When `value` is `true`, the text in the Text object is italicized. When `value` is `false`, the text is roman.

## Properties

### Value — Use italic or roman for text object

`true` or `1` (default) | `false` or `0`

Whether to use italic for the text in the paragraph, specified as a numeric or logical `1` (`true`) or `0` (`false`).

- `true` — Renders text in italic
- `false` — Renders text in roman

### Attributes:

GetAccess	public
SetAccess	public
NonCopyable	true

Data Types: `logical`

### Tag — Tag for this document element

`character vector` | `string scalar`

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

## Examples

### Create a Paragraph with Italic Text

In this example, you italicize the text in a Document object by using an Italic object.

Import the DOM API package so that you do not have to use long, fully qualified names.

```
import mlreportgen.dom.*
```

Create a Document object.

```
d = Document("mydoc", "html");
```

Create a Paragraph object that contains the text that you want to italicize.

```
para = Paragraph("italic text");
```

Italicize the text in para by using an Italic object. Append the italicized paragraph to the document.

```
para.Style = {Italic()};
append(d, para);
```

Close and view the report.

```
close(d);
rptview("mydoc", "html");
```

### Add Italic and Roman Text

In this example, you append italicized and roman text in a `Document` object by using an `Italic` object.

Import the DOM API package so that you do not have to use long, fully qualified names.

```
import mlreportgen.dom.*
```

Create a `Document` object.

```
d = Document("mydoc", "html");
```

Create a `Paragraph` object that contains the text that you want to italicise.

```
para1 = Paragraph("italic text");
```

Italicize the text in `para1` by using an `Italic` object with the value set to `true`. Append the italicized paragraph to the document.

```
para1.Style = {Italic(true)};  
append(d, para1);
```

Create a `Paragraph` object that contains the text that you want in roman font.

```
para2 = Paragraph("roman text");
```

Keep the text in `para2` in roman font by using an `Italic` object with the value set to `false`. Append the paragraph to the document.

```
para2.Style = {Italic(false)};  
append(d, para2);
```

Close and view the report.

```
close(d);  
rptview("mydoc", "html");
```

### See Also

#### Topics

“Report Formatting Approaches” on page 13-17

#### Introduced in R2014b



# mlreportgen.dom.KeepLinesTogether class

**Package:** mlreportgen.dom

Start paragraph on new page if necessary

## Description

Start paragraph on new page if necessary

## Construction

`keepLinesTogetherObj = KeepLinesTogether()` starts a paragraph on a new page if it cannot fit entirely on current page.

`keepLinesTogetherObj = KeepLinesTogether(onoff)` starts paragraph on a new page only if it cannot fit entirely on current page and `onoff` is `true`.

## Input Arguments

### **onoff — Keep paragraph on one page**

logical

Use one of these values:

- `true` (default)
- `false`
- `0`

A setting of `true` (or `1`) starts a paragraph on a new page when it cannot fit entirely on the current page. A setting of `false` (or `0`) allows a paragraph to span two pages when it cannot fit entirely on the current page.

Data Types: logical

## Output Arguments

### **keepLinesTogetherObj — Start paragraph on new page if necessary**

mlreportgen.dom.KeepLinesTogether object

Start paragraph on new page if necessary, represented by an mlreportgen.dom.KeepLinesTogether object.

## Properties

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Value — Keep paragraph lines together**

logical

Possible values are:

- `true` or `1` — Starts paragraph on a new page when it cannot fit entirely on the current page.
- `false` or `0` — Allows the paragraph to span two pages when it cannot fit entirely on the current page.

Data Types: logical

**See Also**

`mlreportgen.dom.PageBreakBefore` | `mlreportgen.dom.KeepWithNext`

**Topics**

“Report Formatting Approaches” on page 13-17

# mlreportgen.dom.KeepWithNext class

**Package:** mlreportgen.dom

Keep paragraph on same page as next

## Description

Keep paragraph on same page as the paragraph that follows it. This format applies to Microsoft Word and PDF documents.

## Construction

`obj = KeepWithNext()` keeps a paragraph on the same page as the paragraph that follows it.

`obj = KeepWithNext(onoff)` keeps a paragraph on the same page as the paragraph that follows it if `onoff` is `true`.

## Input Arguments

**onoff — Keep paragraph on same page as next**

logical

Use one of these values:

- `true` (default)
- `false`
- `1`
- `0`

A setting of `true` (or `1`) keeps a paragraph on the same page as the paragraph that follows it. A setting of `false` (or `0`) allows a paragraph to be on a different page from the paragraph that follows it.

Data Types: logical

## Output Arguments

**keepWithNextObj — Keep paragraph on same page as next**

mlreportgen.dom.KeepWithNext object

Keep paragraph on same page as next, represented by an `mlreportgen.dom.KeepWithNext` object.

## Properties

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Value — Keep paragraph on same page as next paragraph**

logical

Use one of these values:

- `true` or `1` — Keeps a paragraph on the same page as the paragraph that follows it.
- `false` or `0` — Allows a paragraph to be on a different page from the paragraph that follows it.

Data Types: `logical`

**See Also**

`mlreportgen.dom.PageBreakBefore` | `mlreportgen.dom.KeepLinesTogether`

**Topics**

“Report Formatting Approaches” on page 13-17

# mlreportgen.dom.Leader class

**Package:** mlreportgen.dom

Leader character for PDF output

## Description

Create a leader character for PDF documents. A leader is a repeated character that fills out the available space in text.

## Construction

`lead = Leader()` creates a leader object using a dot leader.

`lead = Leader(pattern)` uses the specified leader type.

## Input Arguments

**pattern — Type of leader to use**

'.' (default) | ' ' (space) | 'dots' | 'space'

Type of leader to use, specified as:

- '.' or 'dots' for a dot leader
- ' ' (space character) or 'space' for a space

## Output Arguments

**lead — Leader**

mlreportgen.dom.Leader object

Leader, returned as an mlreportgen.dom.Leader object.

## Properties

**Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**CustomAttributes — Custom attributes of this element**

array of mlreportgen.dom.CustomAttribute objects

Custom attributes of this element, specified as an array of mlreportgen.dom.CustomAttribute objects. Use custom attributes supported by the output format.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Pattern — Type of leader to use**

'.' (default) | ' ' (space) | 'dots' | 'space'

Type of leader to use, specified as:

- '.' or 'dots' for a dot leader
- ' ' (space character) or 'space' for a space

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Style — Format specification**

array of format objects

Format objects that specify the format of a document element.

**StyleName — Style to apply from style sheet**

character vector

Name of the style to apply from the style sheet, specified as a character vector.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Methods

Method	Purpose
clone	Copy this object.

## Examples

### Insert Leaders in a PDF Report

This example uses a dot leader and a space leader in a PDF report.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'pdf');
open(d);

h = Heading1('Cast');
```

```

h.HAlign = 'center';

% Create a leader object l using a space as the leader type
% Append the leader object to a Heading2 paragraph
l = Leader(' ');
h2 = Heading2('Role');
append(h2,l);
append(h2,'Actor');
append(d,h);
append(d,h2);

% Create a leader object dotl using the default leader type of a dot
% Define variables for the content
dotl = Leader();
role = 'Romeo';
actor = 'Leonardo DiCaprio';

% Append the variable text and leader object to a paragraph
p = Paragraph();
append(p,role);
append(p,dotl);
append(p,actor);
append(d,p);

% Repeat, updating variables for each new paragraph
% Insert a clone of the dotl object
role = 'Juliet';
actor = 'Claire Danes';
p = Paragraph();
append(p,role);
append(p,clone(dotl));
append(p,actor);
append(d,p);

role = 'Tybalt';
actor = 'John Leguizamo';
p = Paragraph();
append(p,role);
append(p,clone(dotl));
append(p,actor);
append(d,p);

close(d);
rptview(d.OutputPath);

```

## See Also

mlreportgen.dom.TOC | mlreportgen.dom.Paragraph | mlreportgen.dom.Heading1

## Topics

“Report Formatting Approaches” on page 13-17

## Introduced in R2016a

## mlreportgen.dom.LineBreak class

**Package:** mlreportgen.dom mlreportgen.dom

Line break in a Word or PDF report

### Description

Use an object of `mlreportgen.dom.LineBreak` to insert a line break anywhere in a Microsoft Word or PDF report.

The `mlreportgen.dom.LineBreak` class is a handle class.

### Class Attributes

<code>HandleCompatible</code>	<code>true</code>
<code>ConstructOnLoad</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`lBreak = mlreportgen.dom.LineBreak()` creates a line break object.

## Properties

### Children — Children of this DOM API object

array of DOM API objects

Children of this DOM API object, specified as an array of DOM API objects.

#### Attributes:

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>private</code>
<code>NonCopyable</code>	<code>true</code>

### CustomAttributes — Custom attributes of this element

array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. Use custom attributes supported by the output format.

#### Attributes:

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>
<code>NonCopyable</code>	<code>true</code>



**Id — ID for this DOM API object**

character vector | string scalar

ID for this DOM API object, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Parent — Parent of this DOM API object**

DOM API object

Parent of this DOM API object, specified as a DOM API object.

**Attributes:**

GetAccess	public
SetAccess	private
NonCopyable	true

**Style — Text formatting**

array of format objects

Format objects that specify the format of a document element.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**StyleName — Style to apply from style sheet**

character vector | string scalar

Name of the style to apply from the style sheet, specified as a character vector or string scalar.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Tag — Tag for this DOM API object**

character vector | string scalar

Tag for this DOM API object, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Methods****Public Methods**

Method	Purpose
clone	Copy this object. Use the clone method for a LineBreak object in the same way that you use the clone method for a Paragraph object.

**Examples****Insert a Line Break**

This example shows how to insert a line break in a PDF report by using a `LineBreak` object.

Import the DOM API package so you do not have to use long, fully qualified names.

```
import mlreportgen.dom.*;
```

Create a `Document` object and open it.

```
d = Document("LineBreakReport","pdf");
open(d);
```

Create the first page and add a heading to the object. Create a `Paragraph` object `p`.

```
h = Heading1("Learning to Use Line Breaks");
append(d,h);
p = Paragraph("Use a LineBreak object");
```

Specify `preserve` for the `WhiteSpace` method for the `Paragraph` object to add a line break to it. Append the `LineBreak` handle class to the paragraph.

```
p.WhiteSpace = 'preserve';
append(p,LineBreak);
```

Insert the next line by appending it to the existing `Paragraph` object. Append the paragraph to the `Document`.

```
append(p,"to force a new line in a paragraph");
append(d,p);
```

Close and view the document.

```
close(d);
rptview(d);
```

## **See Also**

[mlreportgen.dom.Paragraph](#) | [mlreportgen.dom.PageBreakBefore](#) |  
[mlreportgen.dom.KeepWithNext](#) | [mlreportgen.dom.KeepLinesTogether](#) |  
[mlreportgen.dom.WidowOrphanControl](#)

## **Topics**

“Report Formatting Approaches” on page 13-17

## **Introduced in R2016b**

## mlreportgen.dom.LineSpacing class

**Package:** mlreportgen.dom

Spacing between lines of paragraph

### Description

Specifies the spacing between lines of a paragraph.

### Construction

`lineSpacingObj = LineSpacing()` specifies line spacing equal to the height of one line at the paragraph font size.

`lineSpacingObj = LineSpacing(multiple)` specifies a line spacing as a multiple of the paragraph text line height (for example, 1.5).

`lineSpacingObj = LineSpacing(spacingHeight)` specifies line spacing as a dimension, for example, '10pt'.

`lineSpacingObj = LineSpacing(spacingHeight, spacingType)` specifies line spacing value and type.

### Input Arguments

#### **multiple** — Multiple of paragraph line height

1 (default) | scalar

Scalar that specifies the line spacing relative to the paragraph text line height.

#### **spacingHeight** — Height of line spacing

character vector

Height of line spacing in the form `valueUnits`, where `Units` is an abbreviation for the units. These abbreviations are valid:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

#### **spacingType** — Line spacing type

'multiple' | 'exact' | 'atleast'

Type of line spacing, specified as one of these values:

- 'multiple' — Value is the spacing in terms of number of lines.

- 'exact' — Value is the exact size of the line spacing.
- 'atleast' — Value is the minimum size of the line spacing (applies only to Microsoft Word)

### Output Arguments

#### **LineSpacingObj** — Spacing between lines of paragraph

mlreportgen.dom.LineSpacing object

Spacing between lines of paragraph, represented by an mlreportgen.dom.LineSpacing object.

### Properties

#### **Id** — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

#### **Tag** — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

#### **Type** — Option to specify type of line spacing height

'multiple' | 'exact' | 'atleast'

Type of line spacing, specified as one of these values:

- 'multiple' — Value is the spacing in terms of number of lines.
- 'exact' — Value is the exact size of the line spacing.
- 'atleast' — Value is the minimum size of the line spacing (applies only to Word)

#### **Value** — Height of line spacing

'lin' (default) | character vector

Height of line spacing in the form valueUnits, where Units is an abbreviation for the units. These abbreviations are valid:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

## **Examples**

### **Create Line Spacing 1.5 Times the Height of the Paragraph Text Lines**

```
p = Paragraph();  
p.Style = {LineSpacing(1.5)};
```

### **See Also**

# mlreportgen.dom.LinkTarget class

**Package:** mlreportgen.dom

Target for internal or external links or image area links

## Description

A target to use for internal and external links and for image area links. You can specify a `LinkTarget` object when you construct an `mlreportgen.dom.InternalLink` or `mlreportgen.dom.ImageArea` object.

## Construction

`targetObj = LinkTarget(name)` creates a link target with the specified name.

### Input Arguments

**name — Name of link target**

character vector | string scalar

Name of a link target, specified as a character vector or string scalar.

---

**Note** To generate a link target name that is valid for all report types, use `mlreportgen.utils.normalizeLinkID`. The generated name conforms to the Microsoft Word limitation on ID length and the PDF requirement that an ID begin with an alphabetic character.

Word replaces spaces in link target names with underscore characters. Avoid spaces in link target names in Word reports.

---

To set up a link target for an external link:

- In a Word report, specify a Word bookmark.
- In an HTML report, specify an HTML named anchor (for example, `<a name='appendix' />`).

### Output Arguments

**targetObj — Link target object**

`mlreportgen.dom.LinkTarget` object

Link target, represented by an `mlreportgen.dom.LinkTarget` object.

## Properties

**CustomAttributes — Custom attributes of document element**

array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this document element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. The custom attributes must be supported by the output format.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Name — Name of link target**

character vector | string scalar

See name input argument.

**Style — Format specification**

array of format objects

Format objects that specify the format of a document element.

**StyleName — Link target style name**

character vector

The style specified by `styleName` must be defined in the template used to create the document element to which this link target is appended.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**IsXRefTarget — Whether this object is target of `mLreportgen.dom.XRef` in PDF report**

false or 0 (default) | true or 1

Whether this object is the target of an `mLreportgen.dom.XRef` object in an `mLreportgen.dom.Document` or `mLreportgen.report.Report` object of type PDF, specified as a numeric or logical 1 (true) or 0 (false). For example see “Use Cross-Reference Elements in a PDF Report” on page 12-146.

**Methods**

Method	Purpose
<code>append</code>	Append content to link target.
<code>clone</code>	Copy link target.
Use <code>LinkTarget.clone</code> in a similar way to how you use <code>Paragraph.clone</code> .	

**Examples**



## Link to Top of a Document

Define a link target at the top of the report and add an internal link to that target.

```
import mlreportgen.dom.*
import mlreportgen.utils.*
d = Document('mydoc', 'pdf');

p = Paragraph('This is my paragraph');
linkID = normalizeLinkID('home');
append(p, LinkTarget(linkID));
append(d, p);

p = Paragraph('This is another paragraph');
p.Style = {PageBreakBefore(true)};
append(d, p);
append(d, InternalLink(linkID, 'Go to Top'));

close(d);
rptview(d);
```

## Compatibility Considerations

### Generate valid link target name for all report types

In a future release, a link target name (ID) generated by `mlreportgen.utils.hash` might not be valid for PDF reports. To ensure that a link target name is valid for all report types, use `mlreportgen.utils.normalizeLinkID` instead of `mlreportgen.utils.hash`.

### See Also

`mlreportgen.dom.ExternalLink` | `mlreportgen.dom.InternalLink` |  
`mlreportgen.dom.XRef` | `mlreportgen.dom.ImageArea` | `mlreportgen.utils.hash` |  
`mlreportgen.utils.normalizeLinkID`

### Topics

“Create Links” on page 13-82

## mlreportgen.dom.ListItem class

**Package:** mlreportgen.dom

Create item for ordered or unordered list

### Description

Specifies an item in an ordered (numbered) or unordered (bulleted) list.

### Construction

`listItemObj = ListItem()` creates an empty list item.

`listItemObj = ListItem(text)` creates a list item using the specified text. The constructor creates a text object and appends the text object to the list item. In Microsoft Word and PDF output, text in a list item is wrapped in a paragraph because Word and PDF do not permit unwrapped text in list items. In HTML output, the text is not wrapped in a paragraph.

`listItemObj = ListItem(text, styleName)` creates a list item using the specified text and applies the specified style.

`listItemObj = ListItem(domObj)` creates a list item and appends the specified document element object to the list item.

`listItemObj = ListItem(domObj, styleName)` creates a list item using the specified document element object and style name.

### Input Arguments

#### **text — Text for list item**

character vector

The constructor creates an `mlreportgen.dom.Text` object for the specified text.

#### **domObj — Document element object**

a DOM object

You can specify a Paragraph object or elements that you can append to a paragraph, including the following kinds of DOM objects:

- `mlreportgen.dom.Text`
- `mlreportgen.dom.Paragraph`
- `mlreportgen.dom.Image`
- `mlreportgen.dom.Table`
- `mlreportgen.dom.FormalTable`
- `mlreportgen.dom.ExternalLink`
- `mlreportgen.dom.InternalLink`
- `mlreportgen.dom.CustomElement`

**styleName — Name of style for list item**

character vector

Name of style to use for the list item, specified as a character vector.

**Output Arguments****ListItemObj — List item**

mlreportgen.dom.ListItem object

List item, represented by an mlreportgen.dom.ListItem object.

**Properties****Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**CustomAttributes — Custom attributes of document element**

array of mlreportgen.dom.CustomAttribute objects

Custom attributes of this document element, specified as an array of mlreportgen.dom.CustomAttribute objects. The custom attributes must be supported by the output format.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Style — Format specification**

array of format objects

Format objects that specify the format of a document element.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**StyleName — This property is ignored**

character vector

This property is ignored.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the

object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Methods

Method	Purpose
<p><code>append</code></p> <p>Use <code>ListItem.append</code> in a similar way as you use <code>Paragraph.append</code>, except you can append different content to a list item than to a paragraph.</p>	<p>Append a character vector or any of these kinds of DOM objects to a list item:</p> <ul style="list-style-type: none"> <li>• <code>mlreportgen.dom.CustomElement</code></li> <li>• <code>mlreportgen.dom.EmbeddedObject</code></li> <li>• <code>mlreportgen.dom.ExternalLink</code></li> <li>• <code>mlreportgen.dom.FormalTable</code></li> <li>• <code>mlreportgen.dom.Image</code></li> <li>• <code>mlreportgen.dom.InternalLink</code></li> <li>• <code>mlreportgen.dom.LineBreak</code></li> <li>• <code>mlreportgen.dom.MATLABTable</code></li> <li>• <code>mlreportgen.dom.NumPages</code></li> <li>• <code>mlreportgen.dom.OrderedList</code></li> <li>• <code>mlreportgen.dom.Page</code></li> <li>• <code>mlreportgen.dom.PageRef</code></li> <li>• <code>mlreportgen.dom.Paragraph</code></li> <li>• <code>mlreportgen.dom.Table</code></li> <li>• <code>mlreportgen.dom.Text</code></li> </ul> <p>In Microsoft Word and PDF output, text in a list item is wrapped in a paragraph because Word and PDF do not permit unwrapped text in list items. In HTML output, the text is not wrapped in a paragraph.</p> <ul style="list-style-type: none"> <li>• <code>mlreportgen.dom.UnorderedList</code></li> </ul>
<p><code>clone</code></p> <p>Use <code>ListItem.clone</code> the same way you use <code>Paragraph.clone</code>.</p>	<p>Clone a list item.</p>

## Examples

### Create List Items for an Ordered List

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);
p = Paragraph('Perform the following steps.');
```

`append(d,p);`

```
step1 = ListItem('Do this step first.');
```

```
textForItem = Text('Next, do this.');
```

```
step2 = ListItem(textForItem);
```

```
procedure = OrderedList();
```

```
append(procedure,step1);
```

```
append(procedure,step2);
```

```
append(d,procedure);
```

```
close(d);
```

```
rptview('test',doctype);
```

## Tips

- Depending on how you create a multilevel list, a sublist can be a child of the parent list or a child of the preceding list item in the parent list. If the sublist is child of a list item in the parent list, the sublist inherits the formatting from the list item. If the sublist is a child of the parent list, the sublist does not inherit the formatting from the previous list item. See “Format a List Item in a Multilevel List” on page 13-59.

## See Also

[mlreportgen.dom.OrderedList](#) | [mlreportgen.dom.UnorderedList](#) | [mlreportgen.dom.ListStyleType](#)

## Topics

“Create and Format Lists” on page 13-51

## mreportgen.dom.ListStyleType class

**Package:** mreportgen.dom

Bullet or number style for HTML and PDF output

### Description

Specifies the bullet or number style for an `mreportgen.dom.ListItem` object for HTML and PDF output.

---

**Note** The default fonts used for PDF output support commonly used bulleted and numbered list item styles. For rarer styles, you must specify a font family that supports the style in the list item's `Style` property or style sheet. You must also configure the DOM API to use the specified font family. See "Configure PDF Fonts" on page 9-32.

---

### Construction

`ListStyleObj = ListStyleType()` specifies a 'disc' style bullet type, which is a filled circle.

`ListStyleObj = ListStyleType(Value)` specifies a bullet or numbering type using any value available for the CSS `list-style-type` property.

### Input Arguments

#### Value — Bullet or number style

'disc' (default) | character vector

Bullet or number style, specified as one of these character vectors.

Style value	Bullet or number type
'disc'	Filled circle (default)
'armenian'	Armenian numbering
'circle'	Open circle
'cjk-ideographic'	Plain ideographic numbers
'decimal'	Number
'decimal-leading-zero'	Number with leading zeroes, i.e., 01, 02, 03, and so on
'georgian'	Georgian numbering
'hebrew'	Hebrew numbering
'hiragana'	Hiragana numbering
'hiragana-iroha'	Hiragana-iroha numbering
'katakana'	Katakana numbering
'katakana-iroha'	Katakana-iroha numbering
'lower-alpha'	Lowercase alphabetic numbering

Style value	Bullet or number type
'lower-greek'	Lowercase Greek alphabetic numbering
'lower-latin'	Lowercase Latin alphabetic numbering
'lower-roman'	Lowercase roman numerals
'none'	No bullet or number
'upper-alpha'	Uppercase alphabetic numbering
'upper-latin'	Uppercase Latin numbering
'upper-roman'	Uppercase roman numerals

## Output Arguments

### ListStyleObj — Bullet or number style for a list item

mlreportgen.dom.ListStyleType object

Bullet or number style for a list item, returned as an mlreportgen.dom.ListStyleType object.

## Properties

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Value — Bullet or number style

'disc' (default) | character vector

Bullet or number style, specified as one of these values.

Style value	Bullet or number type
'disc'	Filled circle (default)
'circle'	Open circle
'cjk-ideographic'	Plain ideographic numbers
'decimal'	Number
'decimal-leading-zero'	Number with leading zeroes, i.e., 01, 02, 03, and so on
'hiragana'	Hiragana numbering

Style value	Bullet or number type
'hiragana-iroha'	Hiragana-iroha numbering
'katakana'	Katakana numbering
'katakana-iroha'	Katakana-iroha numbering
'lower-alpha'	Lowercase alphabetic numbering
'lower-greek'	Lowercase Greek alphabetic numbering
'lower-latin'	Lowercase Latin alphabetic numbering
'lower-roman'	Lowercase roman numerals
'none'	No marker
'upper-alpha'	Uppercase alphabetic numbering
'upper-latin'	Uppercase Latin numbering
'upper-roman'	Uppercase roman numerals

## Examples

### Use Hebrew Numbering in a List for HTML Output

Create a document `mylist` for HTML, the default output type. Add a paragraph to the document and define two list items, giving each the `ListStyleType` property of `'lower-greek'`. Create an ordered list object `procedure` and append the two list items to it, and then append the unordered list to the document.

```
import mlreportgen.dom.*;
d = Document('mylist');
p = Paragraph('This list uses Greek numbering.');
```

```
append(d,p);

step1 = ListItem('Do this step first.');
```

```
step2 = ListItem('Now do this step.');
```

```
step1.Style = {ListStyleType('lower-greek')};
```

```
step2.Style = {ListStyleType('lower-greek')};
```

```
procedure = OrderedList();
```

```
append(procedure,step1);
```

```
append(procedure,step2);
```

```
append(d,procedure);

close(d);
rptview('mylist');
```

### See Also

`mlreportgen.dom.ListItem` | `mlreportgen.dom.OrderedList` | `mlreportgen.dom.UnorderedList`

### Topics

“Create and Format Lists” on page 13-51

“Report Formatting Approaches” on page 13-17



## mlreportgen.dom.LOC class

**Package:** mlreportgen.dom mlreportgen.dom

List of captions

### Description

Use an object of the `mlreportgen.dom.LOC` class to generate a list of captioned report elements. To generate a list of figures with captions or tables with titles, use objects of the `mlreportgen.dom.LOF` and `mlreportgen.dom.LOT` classes, respectively. To generate a list for other types of report elements, such as equations, use an `LOC` object.

Each list item contains the caption of a report element and links to the caption in the report. In a PDF or Microsoft Word report, a list item also includes the page number and a leader that fills the space between the caption and page number.

In PDF and Word reports, the list of captions is placed at the location that you specified in the report generation program. In HTML reports, the list of captions is placed in a side panel and has a title that consists of `List` of followed by the value of the `AutoNumberStreamName` property with the first letter capitalized.

To include report elements in the list:

- 1 Choose a numbering stream name, for example, `equation`. Set the `AutoNumberStreamName` property of the `LOC` object to the numbering stream name.
- 2 Create captions or titles for the report elements using `mlreportgen.dom.Paragraph` objects.
- 3 Associate the `Paragraph` objects with the numbering stream name by using an `mlreportgen.dom.AutoNumber` object.

The way a list is generated depends on the report type.

- PDF — The DOM API generates the list during report generation.
- Word — The DOM API generates a placeholder for the list. To generate the list items, you must update the Word document in your report generation program or in Word. See “Update a Table of Contents or Generated List in a Word Document” on page 3-37.
- HTML — The DOM API generates a placeholder for the list. When the report opens in an HTML browser, the browser generates the list.

---

**Note** You can use an `LOC` object for captions that follow a report element or titles that precede a report element.

---

The `mlreportgen.dom.LOC` class is a `handle` class.

### Class Attributes

<code>ConstructOnLoad</code>	<code>true</code>
<code>HandleCompatible</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`LOCobj = mlreportgen.dom.LOC()` creates an `mlreportgen.dom.LOC` object and sets the `LeaderPattern` property to `'.'`.

`LOCobj = mlreportgen.dom.LOC(leaderPattern)` creates an `mlreportgen.dom.LOC` object and sets the `LeaderPattern` property to the specified leader pattern.

## Properties

### **AutoNumberStreamName — Name of numbering stream**

`' '` (default) | character vector | string scalar

Name of numbering stream, specified as a character vector or string scalar.

### **LeaderPattern — Type of leader**

`'.'` (default) | `'dots'` | `' '` | `'space'`

Type of leader to use between the caption and the page number, specified as one of these character vectors or string scalars:

- `'.'` or `'dots'`
- `' '` or `'space'`

This property applies only to PDF reports. Word reports always have a dots leader. HTML reports do not have a leader.

### **StyleName — Name of stylesheet-defined style for formatting list of captions**

`[]` (default) | character vector | string scalar

Name of the stylesheet-defined style for formatting the list of captions, specified as a character vector or string scalar.

The style specified by the `StyleName` property must be defined in the stylesheet of the document or document part to which this list of captions is appended. The specified style defines the appearance of the list of captions in the output document, except for formats that are specified by the `Style` property of this `LOC` object. The format objects specified by the `Style` property override formats defined by the stylesheet.

### **Style — Formats that define list of captions style**

`{}` (default) | cell array of DOM format objects

Formats that define the style of the list of captions, specified as a cell array of DOM format objects. The formats override the corresponding formats defined by the stylesheet style specified by the `StyleName` property. Formats that do not apply to a list of captions are ignored.

### **CustomAttributes — Custom attributes**

`[]` (default) | array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this document element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. The custom attributes must be supported by the output format.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Methods****Public Methods**

<code>clone</code>	<code>LOCObjCopy = clone(LOCObj)</code> copies (clones) the specified LOC object. Use the <code>clone</code> method of an LOC object in the same way that you use the <code>clone</code> method of an <code>mlreportgen.dom.Paragraph</code> object.
--------------------	--

**Examples****Generate a List of Captions in a Report**

This example generates a list of the captions for example code in a report. You can use the same procedure to generate a list of captions for other report elements. This example identifies the captions to include in the list by associating the captions with the automatic numbering stream that has the name `example`. You can use any name for the numbering stream as long as the name matches the value of the `AutoNumberStream` property of the `mlreportgen.dom.LOC` object that represents the list of captions.

Import the DOM API package so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
```

Create a report.

```
d = Document("DOM Report with List of Captions","pdf");
```

Create a list of captions placeholder and append it to the report.

```
locObj = LOC();
locObj.AutoNumberStreamName = "example";
append(d,locObj);
append(d,PageBreak);
```

Create a paragraph that includes example code.

```
pr1 = Preformatted(" a = 1;" + newline + " b = 2;" + newline + " c = a + b;");
append(d,pr1);
```

Create a paragraph for the example code caption.

```
p1 = Paragraph("Example ");
```

Create an automatic numbering stream with the name example and associate it with the paragraph.

```
append(p1,AutoNumber("example"));
```

Increment the counter for the numbering stream.

```
p1.Style = {CounterInc("example"),WhiteSpace("preserve")};
```

Append the rest of the caption text to the paragraph and append the paragraph to the report.

```
append(p1, ".");
append(p1, " Addition");
append(d,p1);
```

Create another paragraph that contains example code and a caption for the example code. Associate the example numbering stream with the caption and increment the numbering counter.

```
pr2 = Preformatted(" a = 1;" + newline + " b = 2;" + newline + " c = a * b;");
append(d,pr2);
p2 = Paragraph("Example ");
append(p2,AutoNumber("example"));
p2.Style = {CounterInc("example"),WhiteSpace("preserve")};
```

Append the rest of the caption text to the paragraph and append the paragraph to the report.

```
append(p2, ".");
append(p2, " Multiplication");
append(d,p2);
```

Close and view the report.

```
close(d);
rptview(d);
```

Here is the list of captions in the generated report:

<a href="#">Example 1. Addition.....</a>	<a href="#">2</a>
<a href="#">Example 2. Multiplication.....</a>	<a href="#">2</a>

## **See Also**

`mreportgen.report.ListOfCaptions` | `mreportgen.report.ListOfFigures` |  
`mreportgen.report.ListOfTables` | `mreportgen.dom.LOT` | `mreportgen.dom.LOF` |  
`mreportgen.dom.AutoNumber` | `mreportgen.dom.CounterInc`

## **Topics**

“Create a List of the Captions or Titles of Related Report Elements in a Report API Report” on page 3-46

“Update a Table of Contents or Generated List in a Word Document” on page 3-37

“Create a List of Figures, Tables, or Other Report Elements in a DOM API Report” on page 3-45

“Automatically Number Document Content” on page 13-101

## **Introduced in R2020b**

## mlreportgen.dom.LOF class

**Package:** mlreportgen.dom mlreportgen.dom

**Superclasses:** mlreportgen.dom.LOC

List of figures

### Description

Use an object of the `mlreportgen.dom.LOF` class to create a list of figures in a report.

Each list item contains the caption of a figure (image) and links to the caption in the report. In a PDF or Microsoft Word report, a list item also includes the page number and a leader that fills the space between the caption and page number. In a PDF or Word report, the list is located at the point in the report where you append the `LOC` object. In an HTML report, the list is located in a sidebar with the title `List of Figures`.

The way a list is generated depends on the report type.

- PDF — The DOM API generates the list during report generation.
- Word — The DOM API generates a placeholder for the list. To generate the list items, you must update the Word document in your report generation program or in Word. See “Update a Table of Contents or Generated List in a Word Document” on page 3-37.
- HTML — The DOM API generates a placeholder for the list. When the report opens in an HTML browser, the browser generates the list.

To include figures (images) in the list of figures:

- 1 Create captions for the figures using `mlreportgen.dom.Paragraph` objects.
- 2 Associate the `Paragraph` objects with a numbering stream that has the name `figure` by using an `mlreportgen.dom.AutoNumber` object.

The `mlreportgen.dom.LOF` class is a `handle` class.

### Class Attributes

<code>ConstructOnLoad</code>	<code>true</code>
<code>HandleCompatible</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`lofObj = mlreportgen.dom.LOF()` creates an `mlreportgen.dom.LOF` object and sets the `LeaderPattern` property to `'.'`.

`lofObj = mlreportgen.dom.LOF(leaderPattern)` creates an `mlreportgen.dom.LOF` object and sets the `LeaderPattern` property to the specified leader pattern.

## Properties

### AutoNumberStreamName — Name of numbering stream

'figure' (default)

Name of numbering stream, specified as 'figure'. Do not change the value of this property. To create a list of captions using a custom numbering stream name, use an `mlreportgen.dom.LOC` object.

### LeaderPattern — Type of leader

'.' (default) | 'dots' | ' ' | 'space'

Type of leader to use between the caption and the page number, specified as one of these character vectors or string scalars:

- '.' or 'dots'
- ' ' or 'space'

This property applies only to PDF reports. Word reports always have a dots leader. HTML reports do not have a leader.

### StyleName — Name of stylesheet-defined style for the list of figures

[] (default) | character vector | string scalar

Name of the stylesheet-defined style for the list of figures, specified as a character vector or string scalar.

The style specified by the `StyleName` property must be defined in the stylesheet of the document or document part to which this list of captions is appended. The specified style defines the appearance of the list of figures in the output document. Formats specified by the `Style` property override formats defined by the stylesheet.

### Style — Formats that define the list of figures style

{ } (default) | cell array of DOM format objects

Formats that define the style of the list of figures, specified as a cell array of DOM format objects. The formats override the corresponding formats defined by the stylesheet style specified by the `StyleName` property. Formats that do not apply to a list of figures are ignored.

### CustomAttributes — Custom attributes

[] (default) | array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this document element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. The custom attributes must be supported by the output format.

### Parent — Parent of document element

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

### Children — Children of document element

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Methods****Public Methods**

clone	LOFObjCopy = clone(LOFObj) copies (clones) the specified LOF object. Use the clone method of an LOF object in the same way that you use the clone method of an mlreportgen.dom.Paragraph object.
-------	--

**Examples****Generate a List of Figures in a Report**

Create a list of figures as an mlreportgen.dom.LOF object. To include figures in the list:

- Create captions for the figures as mlreportgen.dom.Paragraph objects.
- Associate the Paragraph objects with a numbering stream that has the name figure.

Import the DOM package so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
```

Create a report.

```
d = Document("DOM Report with List of Figures", "docx");
```

Create an LOF object and append it to the report.

```
LOFObj = LOF();
append(d, LOFObj);
append(d, PageBreak);
```

Include an image (figure) in the report.

```
append(d, Image('peppers.png'));
```

Create a paragraph for the figure caption.



```
p1 = Paragraph("Figure ");
```

Create an automatic numbering stream with the name `figure` and associate it with the paragraph.

```
append(p1,AutoNumber("figure"));
```

Increment the counter for the numbering stream.

```
p1.Style = {CounterInc("figure"),WhiteSpace("preserve")};
```

Append the rest of the caption text to the paragraph and append the paragraph to the report.

```
append(p1, ".");
append(p1, " Peppers");
append(d,p1);
```

Include another image in the report and create a caption for the figure. Associate the caption with the figure numbering stream.

```
append(d,Image("b747.jpg"));
p2 = Paragraph("Figure ");
append(p2,AutoNumber("figure"));
p2.Style = {CounterInc("figure"),WhiteSpace("preserve")};
```

Append the rest of the caption text to the paragraph and append the paragraph to the report.

```
append(p2, ".");
append(p2, " Airplane");
append(d,p2);
```

Close and view the report.

```
close(d);
rptview(d);
```

Here is the list of figures in the generated report:

Figure 1. Peppers .....	1
Figure 2. Airplane.....	2

## See Also

[mlreportgen.report.ListOfFigures](#) | [mlreportgen.report.ListOfTables](#) |  
[mlreportgen.report.ListOfCaptions](#) | [mlreportgen.dom.LOT](#) | [mlreportgen.dom.LOC](#)

## Topics

“Create a List of Figures, Tables, or Other Report Elements in a DOM API Report” on page 3-45  
“Automatically Number Document Content” on page 13-101  
“Update a Table of Contents or Generated List in a Word Document” on page 3-37  
“Create a List of Figures or Tables in a Report API Report” on page 3-40

**Introduced in R2020b**

## mlreportgen.dom.LOT class

**Package:** mlreportgen.dom mlreportgen.dom

**Superclasses:** mlreportgen.dom.LOC

List of tables

### Description

Use an object of the `mlreportgen.dom.LOT` class to create a list of tables in a report.

Each list item contains the title of a table and links to the title in the report. In a PDF or Microsoft Word report, a list item also includes the page number and a leader that fills the space between the title and page number. In a PDF or Word report, the list is located at the point in the report where you append the `LOT` object. In an HTML report, the list is located in a sidebar with the title `List of Tables`.

The way a list is generated depends on the report type.

- PDF — The DOM API generates the list during report generation.
- Word — The DOM API generates a placeholder for the list. To generate the list items, you must update the Word document in your report generation program or in Word. See “Update a Table of Contents or Generated List in a Word Document” on page 3-37.
- HTML — The DOM API generates a placeholder for the list. When the report opens in an HTML browser, the browser generates the list.

To include tables in the list of tables:

- 1 Create titles for the tables using `mlreportgen.dom.Paragraph` objects.
- 2 Associate the `Paragraph` objects with a numbering stream that has the name `table` by using an `mlreportgen.dom.AutoNumber` object.

The `mlreportgen.dom.LOT` class is a `handle` class.

### Class Attributes

<code>ConstructOnLoad</code>	<code>true</code>
<code>HandleCompatible</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`LOTObj = mlreportgen.dom.LOT()` creates an `mlreportgen.dom.LOT` object and sets the `LeaderPattern` property to `' '`.

`LOTObj = mlreportgen.dom.LOT(leaderPattern)` creates an `mlreportgen.dom.LOT` object and sets the `LeaderPattern` property to the specified leader pattern.

## Properties

### **AutoNumberStreamName — Name of numbering stream**

'table' (default)

Name of numbering stream, specified as 'table'. Do not change the value of this property. To create a list of titles or captions using a custom numbering stream name, use an `mlreportgen.dom.LOC` object.

### **LeaderPattern — Type of leader**

'.' (default) | 'dots' | ' ' | 'space'

Type of leader to use between the title and the page number, specified as one of these character vectors or string scalars:

- '.' or 'dots'
- ' ' or 'space'

This property applies only to PDF reports. Word reports always have a dots leader. HTML reports do not have a leader.

### **StyleName — Name of stylesheet-defined style for formatting list of tables**

[] (default) | character vector | string scalar

Name of the stylesheet-defined style for formatting the list of tables, specified as a character vector or string scalar.

The style specified by the `StyleName` property must be defined in the stylesheet of the document or document part to which this list of tables titles is appended. The specified style defines the appearance of the list of tables in the output document, except for formats that are specified by the `Style` property of this `LOT` object. The format objects specified by the `Style` property override formats defined by the stylesheet.

### **Style — Formats that define list of tables style**

{ } (default) | cell array of DOM format objects

Formats that define the style of the list of tables, specified as a cell array of DOM format objects. The formats override the corresponding formats defined by the stylesheet style specified by the `StyleName` property. Formats that do not apply to a list of tables are ignored.

### **CustomAttributes — Custom attributes**

[] (default) | array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this document element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. The custom attributes must be supported by the output format.

### **Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

### **Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

## Methods

### Public Methods

clone	LOTObjCopy = clone(LOTObj) copies (clones) the specified LOT object. Use the clone method of an LOT object in the same way that you use the clone method of an mlreportgen.dom.Paragraph object.
-------	--

## Examples

### Generate a List of Tables in a Report

Create a list of tables as an mlreportgen.dom.LOT object. To include tables in the list:

- Create titles for the tables as mlreportgen.dom.Paragraph objects.
- Associate the Paragraph objects with a numbering stream that has the name table.

Import the DOM package so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
```

Create a report.

```
d = Document("DOM Report with List of Tables", "docx");
```

Create a list of tables container and append it to the report.

```
lotObj = LOT();
append(d, lotObj);
append(d, PageBreak);
```

Create a table.

```
t1 = Table(magic(2));
```

Create a paragraph for the table title.

```
p1 = Paragraph("Table ");
```

Create an automatic numbering stream with the name `table` and associate it with the paragraph.

```
append(p1,AutoNumber("table"));
```

Increment the counter for the numbering stream.

```
p1.Style = {CounterInc("table"),WhiteSpace("preserve")};
```

Append the rest of the title text to the paragraph and append the paragraph to the report.

```
append(p1, ".");  
append(p1, " Order 2 Magic Square");  
append(d,p1);
```

Append the table to the report after the table title.

```
append(d,t1);
```

Create another table and a title for the table. Associate the `table` numbering stream with the table title and increment the numbering counter.

```
t2 = Table(magic(3));  
p2 = Paragraph("Table ");  
append(p2,AutoNumber("table"));  
p2.Style = {CounterInc("table"),WhiteSpace("preserve")};
```

Append the rest of the title text to the paragraph and append the paragraph to the report.

```
append(p2, ".");  
append(p2, " Order 3 Magic Square");  
append(d,p2);
```

Append the table to the report after the table title.

```
append(d,t2);
```

Close and view the report.

```
close(d);  
rptview(d);
```

Here is the list of tables in the generated report:

Table 1. Order 2 Magic Square .....	1
Table 2. Order 3 Magic Square .....	1

## **See Also**

`mreportgen.report.ListOfFigures` | `mreportgen.report.ListOfTables` |  
`mreportgen.report.ListOfCaptions` | `mreportgen.dom.LOF` | `mreportgen.dom.LOC`

## **Topics**

“Create a List of Figures, Tables, or Other Report Elements in a DOM API Report” on page 3-45

“Automatically Number Document Content” on page 13-101

“Update a Table of Contents or Generated List in a Word Document” on page 3-37

“Create a List of Figures or Tables in a Report API Report” on page 3-40

## **Introduced in R2020b**

## mlreportgen.dom.MATLABTable class

**Package:** mlreportgen.dom mlreportgen.dom mlreportgen.dom

**Superclasses:** mlreportgen.dom.FormatTable

MATLAB table

### Description

Use an object of the mlreportgen.dom.MATLABTable class to convert a MATLAB table to a DOM table.

The mlreportgen.dom.MATLABTable class is a handle class.

### Class Attributes

HandleCompatible	true
ConstructOnLoad	true

## Creation

### Description

MLTableObj = mlreportgen.dom.MATLABTable(table) creates a table object based on the specified MATLAB table.

MLTableObj = mlreportgen.dom.MATLABTable(table, stylename) creates the table object and sets the StyleName property to stylename. Define the style in the template used to generate the report containing this table.

### Input Arguments

#### table — MATLAB table

MATLAB table

MATLAB table, used to generate the DOM table, specified as a MATLAB table.

## Properties

### BackgroundColor — Background color

[] (default) | character vector | string scalar

Table background color, specified as a character vector or string scalar that consists of a CSS color name or a hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, "#0000ff" specifies blue.



Setting the `BackgroundColor` property adds a `mlreportgen.dom.BackgroundColor` format object to the `Style` property. Setting the `BackgroundColor` property to an empty value removes the object.

Example: "blue"

Example: "#0000ff"

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>
<code>NonCopyable</code>	<code>true</code>

**Body — Table body**

`mlreportgen.dom.TableBody` object

Table body, specified as an `mlreportgen.dom.TableBody` object. The table constructor creates a table body object and assigns it to this property when the formal table is constructed. You cannot subsequently set this property. However, you can append content to the table body and set its properties by using this property.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>private</code>
<code>NonCopyable</code>	<code>true</code>

**Border — Type of border to draw**

"solid" | "single" | "dashed" | "none" | ...

Type of border to draw, specified as one of the values in the table.

Border Value	Description	Supported Output Types
"dashed"	Dashed line	All output types
"dashdotstroked"	Line with alternating diagonal dashes and dot	Word
"dashsmallgap"	Dashed line with a small gap between dashes	Word
"dotted"	Dotted line	All output types
"dotdash"	Line with alternating dots and dashes	Word
"dotdotdash"	Line with alternating double dots and a dash	Word
"double"	Double line	All output types
"doublewave"	Double wavy line	Word
"groove"	3-D effect grooved line	HTML and PDF

Border Value	Description	Supported Output Types
"hidden"	<p>No line</p> <p>The "none" border type also produces no line. However, conflicting borders are handled differently for "hidden" types than for "none" types. The "hidden" border type takes precedence over a conflicting border type. A conflicting border type takes precedence over the "none" type.</p>	HTML and PDF
"inset"	3-D effect line	All output types
"none"	<p>No line</p> <p>The "hidden" border type also produces no line. However, conflicting borders are handled differently for "hidden" types than for "none" types. The "hidden" border type takes precedence over a conflicting border type. A conflicting border type takes precedence over the "none" type.</p>	All output types
"outset"	3-D effect line	All output types
"ridge"	3-D effect ridged line	HTML and PDF
"single"	Single line	Word
"solid"	Single line	HTML and PDF
"thick"	Thick line	Word
"thickthinlargegap"	Dashed line with alternating thick and thin dashes with a large gap	Word
"thickthinmediumgap"	Dashed line with alternating thick and thin dashes with a medium gap	Word
"thickthinsmallgap"	Dashed line with alternating thick and thin dashes with a small gap	Word
"thinthicklargegap"	Dashed line with alternating thin and thick dashes with a large gap	Word
"thinthickmediumgap"	Dashed line with alternating thin and thick dashes with a medium gap	Word

Border Value	Description	Supported Output Types
"thinthicksmallgap"	Dashed line with alternating thin and thick dashes with a small gap	Word
"thinthickthinlargegap"	Dashed line with alternating thin and thick dashes with a large gap	Word
"thinthickthinmediumgap"	Dashed line with alternating thin and thick dashes with a medium gap	Word
"thinthickthinsmallgap"	Dashed line with alternating thin and thick dashes with a small gap	Word
"threedemboss"	Embossed effect line	Word
"threedengrave"	Engraved effect line	Word
"triple"	Triple line	Word
"wave"	Wavy line	Word

**Attributes:**

```

GetAccess                public
SetAccess                public
NonCopyable              true

```

**BorderCollapse — Whether to collapse borders of adjacent cells into single border (HTML only)**

"on" | "off"

Whether to collapse the borders of adjacent cells into a single border, specified as "on" or "off". A value of "on" collapses borders of adjacent cells into a single border. A value of "off" keeps borders of adjacent cells. This property applies only to HTML documents.

**Attributes:**

```

GetAccess                public
SetAccess                public
NonCopyable              true

```

**BorderColor — Border color**

character vector | string scalar

Table border color, specified as a character vector or string scalar that consists of a CSS color name or a hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, "#0000ff" specifies blue.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**BorderWidth — Table border width**

character vector | string scalar

Table border width, specified as a string scalar or character vector that consists of a number followed by an abbreviation of a unit of measurement. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Children — Children of this DOM API object**

array of DOM API objects

Children of this DOM API object, specified as an array of DOM API objects.

**Attributes:**

GetAccess	public
SetAccess	private
NonCopyable	true

**ColSep — Style of line separating columns**

character vector | string scalar

Style of the line that separates the columns of a table, specified as a character vector or string scalar.

See the description of the attribute `Border` for a description of the possible values.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**ColSepColor — Color of line separating columns**

character vector | string scalar

Color of the line that separates columns, specified as a character vector or string scalar that consists of a CSS color name or a hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, "#0000ff" specifies blue.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**ColSepWidth — Width of line separating table columns**

character vector | string scalar

Width of the line separating table columns, specified as a string scalar or character vector that consists of a number followed by an abbreviation of a unit of measurement. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**ColSpecGroups — Properties of group of columns in table**

array of mlreportgen.dom.TableColSpecGroup objects

Width, alignment, and other properties of a group of columns, specified as an array of mlreportgen.dom.TableColSpecGroup objects. The first object applies to the first group of columns, the second object to the second group, and so on. Specify the number of columns that belong to each group by using the Span property of the TableColSpecGroup object. For example, if the first object has a span of 2, it applies to the first two columns. If the second group has a span of 3, it applies to the next three columns, and so on.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**CustomAttributes — Custom attributes for the table**

array of mlreportgen.dom.CustomAttribute objects

Custom attributes for the table, specified as an array of mlreportgen.dom.CustomAttribute objects. The custom attributes must be supported by the output type of the document to which this document element is appended.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**FlowDirection — Text flow direction**

"ltr" | "rtl"

Direction for text to flow, specified as one of these values:

- "ltr" — Text flows from left to right
- "rtl" — Text flows from right to left

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Footer — Table footer**

`mlreportgen.dom.TableFooter` object

Footer for this table, specified as an `mlreportgen.dom.TableFooter` object. The table constructor creates a table footer object and assigns it to this property when the formal table is constructed. You cannot subsequently set this property. However, you can append content to the table body and set its properties by using this property.

**Attributes:**

GetAccess	public
SetAccess	private
NonCopyable	true

**HAlign — Horizontal alignment**

"center" | "left" | "right"

Horizontal alignment of the table, specified as one of these values:

- "center"
- "left"
- "right"

---

**Note** To prevent the overflow of large tables in PDF output, set the `Width` property.

---

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Header — Table header**

`mlreportgen.dom.TableHeader` object

Table header, specified as an `mlreportgen.dom.TableHeader` object. . The table constructor creates a table header object and assigns it to this property when the formal table is constructed. You cannot subsequently set this property. However, you can append content to the table body and set its properties by using this property.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>private</code>
<code>NonCopyable</code>	<code>true</code>

**HeaderRule — Horizontal rule for heading**

`mlreportgen.dom.HorizontalRule` object

Horizontal rule for the heading, specified as an `mlreportgen.dom.HorizontalRule` object.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>
<code>NonCopyable</code>	<code>true</code>

**Id — ID for this DOM API object**

character vector | string scalar

ID for this DOM API object, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>
<code>NonCopyable</code>	<code>true</code>

**NColumns — Number of columns**

integer

Number of columns, specified as an integer.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>private</code>
<code>NonCopyable</code>	<code>true</code>

**OuterLeftMargin — Left margin of table**

character vector | string scalar

Left margin of the table, specified as a string scalar or character vector that consists of a number followed by an abbreviation of a unit of measurement. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters

- pc — picas
- pt — points

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Parent — Parent of this DOM API object**

DOM API object

Parent of this DOM API object, specified as a DOM API object.

**Attributes:**

GetAccess	public
SetAccess	private
NonCopyable	true

**RowNamesRule — Whether to enable row names column**

logical

Whether to enable the first column that contains the row names, specified as logical. The MATLAB table object must define the row names.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**RowSep — Style of lines separating rows**

character vector | string scalar

Style of the lines that separate the rows of the table, specified as a character vector or string scalar.

See the description of the `Border` property for a description of the possible values.**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**RowSepColor — Color of lines separating table rows**

character vector | string scalar

Color of the lines that separate table rows, specified as a character vector or string scalar that consists of a CSS color name or a hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, "#0000ff" specifies blue.



**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**RowSepWidth — Width of lines separating table rows**

character vector | string scalar

Width of the lines that separate the table rows, specified as a string scalar or character vector that consists of a number followed by an abbreviation of a unit of measurement. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Style — Format for table**

array of format objects

Formats that define the style of this table, specified as a cell array of DOM format objects. The formats override the corresponding formats defined by the stylesheet style specified by the `StyleName` property.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**StyleName — Style in document or document part style sheet**

character vector | string scalar

Name of a style specified in the style sheet of the document or document part to which this table is appended, specified as a character vector or string scalar.

The style specified by the `StyleName` property must be defined in the stylesheet of the document or document part to which this table is appended. The specified style defines the appearance of the table in the output document, except for formats that are specified by the `Style` property of this `MATLABTable` object. The format objects specified by the `Style` property override formats defined by the stylesheet.

You can set the `StyleName` property of any formal table section. Setting `StyleName` overrides the style specified by the formal table itself. However, if you do this for a Word document, you must explicitly specify the width of each column in a section to ensure that all sections have the same

width. Word, unlike HTML and PDF, has no built-in support for formal tables. To handle this, the DOM interface represents a formal table as three tables, one for each section, embedded in a 3-by-1 table.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**TableEntriesStyle — Style to use for table entries**

cell array of format objects

Style to use for the table entries, specified as a cell array of format objects.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**TableEntriesInnerMargin — Inner margin for table entries**

character vector | string scalar

Inner margin for the table entries, specified as a string scalar or character vector that consists of a number followed by an abbreviation of a unit of measurement. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Tag — Tag for this DOM API object**

character vector | string scalar

Tag for this DOM API object, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Width — Table width**

character vector | string scalar

Percentage of the page width, such as "100%", or a number followed by an abbreviation of a unit of measurement, specified as a string scalar or character vector. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

For Word report, the width is calculated as a percentage of the page width minus the margins.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Methods****Public Methods**

Method	Purpose
append	Append a row of table entries to a table. Use the append method of a MATLABTable object in the same way as you use the append method of an mlreportgen.dom.TableRow object.
clone	Copy the table. Use the clone method of a MATLABTable object in the same way as you use the clone of an mlreportgen.dom.Paragraph object.

**Examples****Create a Table from a MATLAB Table**

This example creates a DOM table from a MATLAB table that has row names. The example generates two reports that include the DOM table. In the first report, the row names column of the table does not have a label. In the second report, the row names column has a label.

Create a MATLAB table that shows the age, weight, and height of patients. Use the RowNames option to identify each row by the last name of the patient.

```
LastName = {'Sanchez'; 'Johnson'; 'Lee'; 'Diaz'; 'Brown'};
Age = [38; 43; 38; 40; 49];
Weight = [176; 163; 131; 133; 119];
Height = [71; 69; 64; 67; 64];
mltable = table(Age, Weight, Height, 'RowNames', LastName)
```

```
mltable=5x3 table
           Age      Weight      Height
           ---      ---      ---
Sanchez    38       176        71
Johnson   43       163        69
Lee        38       131        64
Diaz       40       133        67
Brown      49       119        64
```

The MATLAB table has five rows and three columns. The row names are not part of the table. They are stored in a property of the table.

Create an `mlreportgen.dom.MATLABTable` object from the MATLAB table.

```
import mlreportgen.dom.*
mltableObj = MATLABTable(mltable);
```

Create a document and append the `MATLABTable` object to the document. Close and view the document.

```
d = Document('MyMATLABTable1','docx');
append(d,mltableObj);
close(d);
rptview(d);
```

Here is the table in the generated report:

	Age	Weight	Height
<b>Sanchez</b>	38	176	71
<b>Johnson</b>	43	163	69
<b>Lee</b>	38	131	64
<b>Diaz</b>	40	133	67
<b>Brown</b>	49	119	64

The DOM table is a formal table, which has a header and a body. The table body has five rows and four columns. The first column consists of the MATLAB table row names.

Generate the report again, this time with a label for the column of row names. To specify the label, replace the empty text in the first entry of the table header row with the label text. To draw a line under the label, set the `RowNamesRule` property of the `MATLABTable` object to `true`.

```
LastName = {'Sanchez'; 'Johnson'; 'Lee'; 'Diaz'; 'Brown'};
Age = [38;43;38;40;49];
Weight = [176;163;131;133;119];
Height = [71;69;64;67;64];
mltable = table(Age,Weight,Height,'RowNames',LastName);
import mlreportgen.dom.*
mltableObj = MATLABTable(mltable);
th = mltableObj.Header;
thentry11 = entry(th,1,1);
thentry11.Children(1).Children(1).Content = 'Names';
mltableObj.RowNamesRule = true;
d = Document('MyMATLABTable2','docx');
```

```
append(d,mltableObj);  
close(d);  
rptview(d);
```

Here is the table in the generated report:

<b>Names</b>	<b>Age</b>	<b>Weight</b>	<b>Height</b>
<b>Sanchez</b>	38	176	71
<b>Johnson</b>	43	163	69
<b>Lee</b>	38	131	64
<b>Diaz</b>	40	133	67
<b>Brown</b>	49	119	64

## **See Also**

`mlreportgen.dom.FormatTable | append`

## **Topics**

“Create Tables from MATLAB Tables” on page 13-68

**Introduced in R2016b**

## mlreportgen.dom.MessageDispatcher class

**Package:** mlreportgen.dom

DOM message dispatcher

### Description

Dispatcher for document generation status messages.

---

**Note** When you create a message dispatcher, the DOM API keeps the dispatcher until the end of the current MATLAB session. Delete message event listeners to avoid duplicate reporting of message objects during a MATLAB session.

---

### Properties

#### Filter — Message filter

character vector

(Read-only) The value of this property is a filter that determines the types of messages the dispatcher dispatches. You can control which types of messages are dispatched by setting the properties of the filter.

#### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

#### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Methods

Method	Purpose
dispatch	Dispatch a document generation status message
mlreportgen.ppt.MessageDispatcher.getTheDispatcher	Get the message dispatcher

### Examples

## Add and Dispatch a Progress Message

This example shows how to add a progress message to display when generating a report.

Add a dispatcher and listener to the report.

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);
d.Tag = 'My report';
    dispatcher = MessageDispatcher.getTheDispatcher;
l = addlistener(dispatcher,'Message', ...
    @(src,evtdata) disp(evtdata.Message.formatAsText));

open(d);
dispatch(dispatcher,ProgressMessage('starting chapter',d));
p = Paragraph('Chapter ');
p.Tag = 'chapter title';
append(d,p);

close(d);
rptview('test',doctype);
```

Delete the listener to avoid duplicate reporting of message objects during a MATLAB session.

```
delete(l);
```

Check the progress messages in the MATLAB Command Window. The starting chapter message appears, in addition to the predefined DOM progress messages.

## See Also

[dispatch](#) | [mlreportgen.dom.MessageDispatcher.getTheDispatcher](#) | [mlreportgen.dom.MessageEventData](#) | [mlreportgen.dom.MessageFilter](#)

## Topics

“Display Progress and Debugger Messages” on page 13-115

## mlreportgen.dom.MessageEventData class

**Package:** mlreportgen.dom

Holds message triggering message event

### Description

Contains the message that triggered a message event.

### Construction

`messageEventDataObj = MessageEventData(msg)` creates a message event data object that contains a DOM message (for example, a message of type `mlreportgen.dom.ProgressMessage`).

The DOM message dispatcher attaches an object of this type to a message event when it dispatches a message. This enables message event listeners to retrieve the dispatched message. You need to create instances of this type only if you want to create your own message dispatcher.

### Input Arguments

#### **msg — Message object**

message object

A message object, such as an `mlreportgen.dom.ProgressMessage` object, that triggers a message event.

### Output Arguments

#### **messageEventDataObj — Holds message triggering message event**

`mlreportgen.dom.MessageEventData` object

Container for message triggering message event data, represented by an `mlreportgen.dom.MessageEventData` object.

### Properties

#### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

#### **Message — Message object (read-only)**

message object

The value of this read-only property is a DOM message object, such as an `mlreportgen.dom.ProgressMessage` object, that triggers a message event.

#### **Tag — Tag for this document element**

character vector | string scalar



Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Examples

### Capture Message Event Data

When you add a dispatcher, the DOM API creates the evtdata object, which is an `mlreportgen.dom.MessageEventData` object.

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test', doctype);
d.Tag = 'My report';

dispatcher = MessageDispatcher.getTheDispatcher;
l = addlistener(dispatcher, 'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsText));

open(d);
dispatch(dispatcher, ProgressMessage('starting chapter', d));
p = Paragraph('Chapter ');
p.Tag = 'chapter title';
p.Style = { CounterInc('chapter'), CounterReset('table'), WhiteSpace('pre') };
append(p, AutoNumber('chapter'));
append(d, p);

close(d);
rptview('test', doctype);
```

Delete the listener to avoid duplicate reporting of message objects during a MATLAB session.

```
delete(l);
```

## See Also

### Topics

“Display Progress and Debugger Messages” on page 13-115

## mlreportgen.dom.MessageFilter class

**Package:** mlreportgen.dom

Filter to control message dispatcher

### Description

Filter for messages dispatched by the message dispatcher.

### Properties

#### **DebugMessagePass — Pass or block debug messages**

logical value

- `true`— Pass debug messages.
- `false`— Block debug messages.

Data Types: `logical`

#### **ErrorMessagePass — Pass or block error messages**

logical value

- `true`— Pass error messages.
- `false`— Block error messages.

Data Types: `logical`

#### **GlobalFilter — Pass or block all messages**

logical value

- `true`— Pass all messages.
- `false`— Block all messages.

Data Types: `logical`

#### **ProgressMessagePass — Pass or block progress messages**

logical value

- `true`— Pass progress messages.
- `false`— Block progress messages.

Data Types: `logical`

#### **GlobalFilter — Pass or block all messages**

logical value

- `true`— Pass all messages.
- `false`— Block all messages.

Data Types: `logical`

**SourceFilter — Pass messages only for this DOM object**

DOM object

Pass messages only from the specified DOM object if the messages meet the other filter conditions specified by this MessageFilter object.

**See Also**

dispatch

**Topics**

“Display Progress and Debugger Messages” on page 13-115

## mlreportgen.dom.Number class

**Package:** mlreportgen.dom

Number to include as formatted text in a document

### Description

Use an object of the mlreportgen.dom.Number class to convert a number to formatted text that you can include in a document.

The conversion uses the first of these format specifications that it finds:

- 1 The specification in an mlreportgen.dom.NumberFormat object in the Style property of the Number object
- 2 The specification in a NumberFormat object in the Style property of an element, such as a paragraph, list, or table, that contains the Number object
- 3 The default specification set by mlreportgen.dom.setDefaultNumberFormat

If the conversion does not find a format specification, the conversion uses the maximum number of digits needed to represent the number accurately.

The mlreportgen.dom.Number class is a handle class.

### Class Attributes

ConstructOnLoad	true
HandleCompatible	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

numberObj = mlreportgen.dom.Number() creates an empty Number object. Use the Value property to specify a number to convert to formatted text.

numberObj = mlreportgen.dom.Number(value) creates a Number object with the Value property set to value.

numberObj = mlreportgen.dom.Number(value, stylename) creates a Number object with the Value property set to value and the StyleName property set to stylename.

### Properties

#### Value — Number to convert to formatted text

0 (default) | scalar

Number to convert to formatted text, specified as a scalar. Complex numbers are not supported.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | logical

**StyleName — Name of stylesheet-defined style for converted number**

[] (default) | character vector | string scalar

Name of the stylesheet-defined style for this number when converted to text, specified as a character vector or string scalar.

The style specified by the `StyleName` property must be defined in the stylesheet of the document or document part to which this `Number` object is appended. The specified style defines the appearance of the textual representation of the number in the output document. Formats specified by the `Style` property override formats defined by the stylesheet.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Style — Formats that define style of converted number**

cell array of DOM format objects

Formats that define the style of this number when converted to text, specified as a cell array of DOM format objects. Use an `mlreportgen.dom.NumberFormat` object to specify the precision of the converted number. The formats override the corresponding formats defined by the stylesheet style specified by the `StyleName` property.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**CustomAttributes — Custom attributes**

[] | cell array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of the document element that contains the converted number, specified as a cell array of `mlreportgen.dom.CustomAttribute` objects. The output format must support the custom attributes.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Parent — Parent of this number object**

document element object

Parent of this number object, specified as a document element object.

**Attributes:**

GetAccess	public
SetAccess	private
NonCopyable	true

**Children — Ignored**

document element object

This property is ignored.

**Attributes:**

GetAccess	public
SetAccess	private
NonCopyable	true

**Tag — Tag for this DOM API object**

character vector | string scalar

Tag for this DOM API object, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Id — ID for this DOM API object**

character vector | string scalar

ID for this DOM API object, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

## Methods

### Public Methods

clone	numObjCopy = clone(numObj) copies the specified Number object. Use the clone method of a Number object in the same way that you use the clone method of an mlreportgen.dom.Paragraph object.
toString	Convert number to formatted text

## Examples

### Format a Number in a Report

Represent a number as a Number object. Use a NumberFormat object to specify four digits of precision after the decimal point.

```
import mlreportgen.dom.*

rpt = Document("Report with NumberFormat", "pdf");

n = Number(pi);
n.Style = [n.Style {NumberFormat("%0.4f")}];
append(rpt,n);

close(rpt);
rptview(rpt);
```

### See Also

mlreportgen.dom.NumberFormat | mlreportgen.dom.getDefaultNumberFormat | mlreportgen.dom.setDefaultNumberFormat

### Topics

“Format Numbers in a Table” on page 17-112

“Format Numbers” on page 13-43

### Introduced in R2021a

## mlreportgen.dom.NumberFormat class

**Package:** mlreportgen.dom

Number formatting

### Description

Use an object of the `mlreportgen.dom.NumberFormat` class to specify the formatting of numbers in reports. To specify the formatting of one number, represent the number as an object of the `mlreportgen.dom.Number` class and include the `NumberFormat` object in the `Style` property of the `Number` object. To format all numbers in a document element, such as a paragraph, list, or table, include the `NumberFormat` object in the `Style` property of the object that represents the element.

The `mlreportgen.dom.NumberFormat` class is a `handle` class.

### Class Attributes

<code>ConstructOnLoad</code>	<code>true</code>
<code>HandleCompatible</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`numberFormatObj = mlreportgen.dom.NumberFormat()` creates a `NumberFormat` object. Set the `Value` property to the format specification.

`numberFormatObj = mlreportgen.dom.NumberFormat(value)` creates a `NumberFormat` object and sets the `Value` property to the format specified by `value`.

## Properties

### Value — Format specification

character vector | string scalar

Format specification, specified as a character vector or string scalar. The specification must be a valid format specification for the `sprintf` function and use one of these operators:

- `%f`
- `%e`
- `%E`
- `%g`
- `%G`

Example: `"%0.2f"`



**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Tag — Tag for this DOM API object**

character vector | string scalar

Tag for this DOM API object, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Id — ID for this DOM API object**

character vector | string scalar

ID for this DOM API object, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Examples****Format a Number in a Report**

Represent a number as a Number object. Use a NumberFormat object to specify four digits of precision after the decimal point.

```
import mlreportgen.dom.*

rpt = Document("Report with NumberFormat", "pdf");

n = Number(pi);
n.Style = [n.Style {NumberFormat("%0.4f")}];
append(rpt,n);

close(rpt);
rptview(rpt);
```

**See Also**

`sprintf` | `mlreportgen.dom.getDefaultNumberFormat` |  
`mlreportgen.dom.setDefaultNumberFormat` | `mlreportgen.dom.Number`

**Topics**

“Format Numbers in a Table” on page 17-112

“Format Numbers” on page 13-43

**Introduced in R2021a**

# mlreportgen.dom.NumPages class

**Package:** mlreportgen.dom

Create placeholder for number of document pages

## Description

Create a placeholder for the number of pages in a document. This object applies only to Word and PDF output. For Word output, opening a Word document causes Word to replace this object with the number of pages in the document. For PDF output, the DOM API replaces this object with the total number of pages when writing the document.

## Construction

`num = NumPages()` creates an object for the total number of pages in the report.

## Output Arguments

**num — Total number of pages**

mlreportgen.dom.NumPages object

Total number of pages, returned as an mlreportgen.dom.NumPages object.

## Properties

**Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**CustomAttributes — Custom attributes of this element**

array of mlreportgen.dom.CustomAttribute objects

Custom attributes of this element, specified as an array of mlreportgen.dom.CustomAttribute objects. Use custom attributes supported by the output format.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Style — Format specification

array of format objects

Format objects that specify the format of a document element.

### StyleName — Style to apply from style sheet

character vector

Name of the style to apply from the style sheet, specified as a character vector.

## Methods

Method	Purpose
clone	Copy this object.

## Examples

### Insert Total Number of Pages

This example inserts the total number of document pages in a page footer. Use this class to display the current page number along with the total number of pages, such as Page 1 of 3.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'docx');
open(d);

% Create page footer
footer = DOCXPageFooter('default');
d.CurrentPageLayout.PageFooters = footer;

% Define page number and add to footer.
d.CurrentPageLayout.FirstPageNumber = 1;
t = Text('Page ');
t.WhiteSpace = 'preserve';
t1 = Text(' of ');
t1.WhiteSpace = 'preserve';
pageinfo = Paragraph();
pageinfo.HAlign = 'center';
append(pageinfo,t);
append(pageinfo,Page());
append(pageinfo,t1);
append(pageinfo,NumPages());
append(footer,pageinfo);

% Create several pages.
p = Paragraph('Hello World');
```

```
append(d,p);  
p = Paragraph('Another page');  
p.Style = {PageBreakBefore(true)};  
append(d,p);  
append(d,clone(p));  
  
close(d);  
rptview(d.OutputPath);
```

## See Also

mlreportgen.dom.Page | mlreportgen.dom.Text | mlreportgen.dom.Paragraph

## Topics

“Report Formatting Approaches” on page 13-17

## Introduced in R2016a

## mlreportgen.dom.OPCPart class

**Package:** mlreportgen.dom

Document part to include in OPC package

### Description

Document part to include in an OPC package.

### Construction

`opcPartObj = OPCPart()` creates an empty OPC part.

`opcPartObj = OPCPart(name, sourcePath)` creates a part having the specified name whose source file is located at the specified path. Appending the part to a document using the `Document.package` method causes a copy of the source file to be inserted in the document package at the location specified by the part name.

### Input Arguments

#### **name — Name of part**

character vector

Name of part, specified as a character vector.

#### **sourcePath — Path of source file for part**

character vector

Path of source file for part, specified as a character vector.

### Output Arguments

#### **opcPartObj — OPC part**

mlreportgen.dom.OPCPart object

OPC part, represented by an mlreportgen.dom.OPCPart object.

### Properties

#### **ContentType — Content type of part**

character vector

Specifies the content type, using a file extension. For a list of file content types, see [https://en.wikipedia.org/wiki/Open\\_Packaging\\_Conventions](https://en.wikipedia.org/wiki/Open_Packaging_Conventions).

If you do not set this property and the part is one of the types listed below, the DOM interface sets the content type when you append the part to a document.

File Type	File Extension
Windows bitmap	.bmp
Cascading style sheet	.css
Plain text	.txt
Icon	.cur
Windows metafile	.emf
Encapsulated PostScript	.eps
GIF image	.gif
HTML	.html
JPEG image	.jpe
JPEG image	.jpeg
JPEG	.jpg
JavaScript	.js
JavaScript object Notation	.json
PNG image	.png
PSD	.psd
Rich Text Format	.rtf
Scalable Vector Graphics	.svg
TIFF image	.tif
TIFF image	.tiff
Truetype font	.ttf

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Name — Path of part**

character vector

Path of this part relative to the root of the package, specified as a character vector. For example, to add an image named `myimage.jpg` to a document images folder, specify the path as  `'/images/myimage.jpg'`. Specify Name using only ASCII characters.

For information about OPC part names, see [https://en.wikipedia.org/wiki/Open\\_Packaging\\_Conventions](https://en.wikipedia.org/wiki/Open_Packaging_Conventions).

**RelatedPart — Path name of part to which specified part is related**

character vector

For information about OPC part names, see [https://en.wikipedia.org/wiki/Open\\_Packaging\\_Conventions](https://en.wikipedia.org/wiki/Open_Packaging_Conventions).

**RelationshipID — Relationship ID**

character vector

For information about OPC relationship IDs, see [https://en.wikipedia.org/wiki/Open\\_Packaging\\_Conventions](https://en.wikipedia.org/wiki/Open_Packaging_Conventions).

### RelationshipType — Relationship type

character vector

Specifies a relationship type, using a file extension. For a list of file content types, see [https://en.wikipedia.org/wiki/Open\\_Packaging\\_Conventions](https://en.wikipedia.org/wiki/Open_Packaging_Conventions).

If you do not set this property and the part is one of these types, the DOM interface sets the content type when you append the part to a document.

File Type	File Extension
Windows bitmap	.bmp
Cascading style sheet	.css
Plain text	.txt
Icon	.cur
Windows metafile	.emf
Encapsulated PostScript	.eps
GIF image	.gif
HTML	.html
JPEG image	.jpe
JPEG image	.jpeg
JPEG	.jpg
JavaScript	.js
JavaScript object Notation	.json
PNG image	.png
PSD	.psd
Rich Text Format	.rtf
Scalable Vector Graphics	.svg
TIFF image	.tif
TIFF image	.tiff
Truetype font	.ttf

### SourceFilePath — Source file path

character vector

Source file path, specified as a character vector.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the



object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Examples

### Add a File to an OPC Package

This code inserts a copy of the `data.json` file in the `data` subfolder of the `mydoc` package. This example assumes that there is a `data.json` file in the current folder.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'html');
package(d, OPCPart('/data/data.json', 'data.json'));
close(d);
```

### See Also

package

### Topics

“Output Types and Report Generator Packages” on page 13-14

## mlreportgen.dom.OrderedList class

**Package:** mlreportgen.dom

Create ordered list

### Description

Create an ordered (numbered) list.

### Construction

`orderedListObj = OrderedList()` creates an empty ordered list.

`orderedListObj = OrderedList(items)` creates an ordered list of the specified items.

### Input Arguments

#### **items — Content to include in ordered list**

one-dimensional array of doubles | one-dimensional array of character vectors | one-dimensional categorical array | one-dimensional cell array

Content to include in an ordered list, specified as a:

- One-dimensional array of doubles
- One-dimensional array of character vectors
- One-dimensional categorical array
- One-dimensional cell array

The cell array can contain a combination of the following:

- A character vector
- A number
- A Boolean value
- One of the following DOM objects:
  - `mlreportgen.dom.Text`
  - `mlreportgen.dom.Paragraph`
  - `mlreportgen.dom.ExternalLink`
  - `mlreportgen.dom.InternalLink`
  - `mlreportgen.dom.Table`
  - `mlreportgen.dom.Image`
  - `mlreportgen.dom.CustomElement`
- Horizontal one-dimensional array (for a sublist)

## Output Arguments

### **orderedListObj** — Ordered list

mlreportgen.dom.OrderedList object

An ordered list containing the specified list items.

## Properties

### **CustomAttributes** — Custom attributes of document element

array of mlreportgen.dom.CustomAttribute objects

Custom attributes of this document element, specified as an array of mlreportgen.dom.CustomAttribute objects. The custom attributes must be supported by the output format.

### **Id** — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **Style** — Format specification

array of format objects

Format objects that specify the format of a document element.

### **Stylename** — List style name

character vector

The style specified by Stylename must be defined in the template used to create the document element to which you append this list.

### **Tag** — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Methods

Method	Purpose
append	Append items to this list.
clone	Copy the list.
Use the OrderedList.clone method similar to how you use Paragraph.clone.	

## Examples

### Create an Ordered List

```
import mlreportgen.dom.*;
d = Document('test','html');

ol = OrderedList({Text('a'), 'b', 1,...
    {'c', Paragraph('d')}});
append(d,ol);

close(d);
rptview('test','html');
```

## Compatibility Considerations

### Default Indentation for Ordered List in Word Report

*Behavior changed in R2019b*

Starting in R2019b, the default indentation for an ordered list in a Microsoft Word report is 0.25 inches. In previous releases, by default, an ordered list was unindented in a Word report. In MATLAB R2019b or a later release, to generate an unindented ordered list, set the outer margin of the list to 0.25 inches.

```
ol = OrderedList({'a', 'b', 'c'});
ol.Style = {mlreportgen.dom.OuterMargin('0.25in','0in')};
```

Specify 0.25 inches, not 0 inches, to allow for the 0.25 inches required for the list item numbers.

### See Also

`mlreportgen.dom.UnorderedList` | `mlreportgen.dom.ListItem`

### Topics

“Create and Format Lists” on page 13-51

# mlreportgen.dom.OuterMargin class

**Package:** mlreportgen.dom

Margin between bounding box and its surroundings

## Description

Specifies the margin between the bounding box of an object and adjacent document objects. A bounding box of an object includes the border of the object (if it has a border), the inner margin, and the object content.

## Construction

`marginObj = OuterMargin()` creates an unspecified margin between the bounding box of an object and its surroundings.

`marginObj = OuterMargin(all)` creates the specified margin on all sides between the bounding box of an object and its surroundings.

`marginObj = OuterMargin(left, right)` creates the specified margins between the left and right sides of the bounding box of an object and its surroundings.

`marginObj = OuterMargin(left, right, top, bottom)` creates the specified margins between sides of the bounding box of an object and its surroundings.

## Input Arguments

### **all** — Outer margin size on all sides

character vector

Margin on all sides between the bounding box of an object and its surroundings in the form `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

### **left** — Outer left margin size

character vector

Left margin between the bounding box of an object and its surroundings. See the `all` argument for description of valid values.

### **right** — Outer right margin size

character vector

Right margin between the bounding box of an object and its surroundings. See the `all` argument for description of valid values.

**top — Outer top margin size**

character vector

Top margin between the bounding box of an object and its surroundings. See the `all` argument for description of valid values. Word reports ignore the Top margin setting of a table.

**bottom — Outer bottom margin size**

character vector

Bottom margin between the bounding box of an object and its surroundings. See the `all` argument for description of valid values. Word reports ignore the Bottom margin setting of a table.

**Output Arguments****marginObj — Margin between bounding box and surroundings**

`mlreportgen.dom.OuterMargin` object

An `mlreportgen.dom.OuterMargin` object specifying the margin between bounding box and surroundings.

**Properties****Bottom — Size of bottom margin**

character vector

Bottom margin in the form `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- `+`
- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

---

**Note** Tables in Word reports ignore the `Bottom` property. To control the spacing between a table and the succeeding content, insert an invisible table or paragraph below the table. For example, to create two successive tables 1 inch apart, insert an invisible 1-inch high table. Alternatively, insert an empty paragraph with a setting for line height setting of 0 and a setting for spacing before of 1 inch.

---

**Left — Size of left margin**

character vector

Left margin size. See the `Bottom` property for description of valid values.

**Right — Size of right margin**

character vector

Right margin size. See the `Bottom` property for description of valid values.

### Top — Size of top margin

character vector

Top margin size. See the `Bottom` property for description of valid values.

---

**Note** Tables in Word reports ignore the `Top` property. To control the spacing between a table and the preceding content, insert an invisible table or paragraph above the table. For example, to create two successive tables 1 inch apart, insert an invisible 1-inch high table. Alternatively, insert an empty paragraph with a setting for line height setting of 0 and a setting for spacing before of 1 inch.

---

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Examples

### Add Margins to Paragraph That Has a Border

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);

p = Paragraph('Hello World');
p.Style = {Border('solid','Red'), ...
    HAlign('center'),...
    OuterMargin('0pt','0pt','0pt','24pt')};
append(d,p);

p = Paragraph('Greetings from MATLAB');
p.Style = {Border('solid','green'), ...
    HAlign('center')};
append(d,p);

p = Paragraph('End of report');
p.Style = {Border('solid','blue'),...
    HAlign('center'),...
    OuterMargin('0pt','0pt','0pt','12pt')};
append(d,p);
```

```
close(d);  
rptview('test',doctype);
```

### **See Also**

mlreportgen.dom.InnerMargin

### **Topics**

“Report Formatting Approaches” on page 13-17



# mlreportgen.dom.OutlineLevel class

**Package:** mlreportgen.dom

Level of paragraph in outline

## Description

Specifies the level of a paragraph in an automatically generated outline. This class is intended for Microsoft Word reports, because HTML does not support displaying paragraphs in a table of contents.

## Construction

`outlineLevelObj = OutlineLevel()` sets the outline level of this paragraph to 1. This causes the content of the paragraph to appear at the top level in an automatically generated outline (for example, a table of contents).

`outlineLevelObj = OutlineLevel(level)` sets the paragraph to the specified outline level.

## Input Arguments

### Level — Specify the level of a paragraph in an outline

integer

Outline level for a paragraph, specified as a positive integer, from 1 to 9.

Data Types: int16

## Output Arguments

### outlineLevelObj — Level of paragraph in outline

mlreportgen.dom.OutlineLevel object

Level of paragraph in outline, represented by an mlreportgen.dom.OutlineLevel object.

## Properties

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the

object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### **Value — Specify the level of a paragraph in an outline**

integer

Outline level for a paragraph, specified as a positive integer, from 1 to 9.

Data Types: int16

## **Examples**

### **Add a Table of Contents**

Add an automatically generated table of contents and set the outline level of the “Glossary” paragraph so that the paragraph appears at the top level of the table of contents. This example uses the default DOM Word template.

Create a document and document part for the table of contents. The document part uses the ReportTOC building block from the default DOM Word template.

```
import mlreportgen.dom.*
d = Document('tocDoc','docx');
open(d);

dp = DocumentPart(d,'ReportTOC');
append(d,dp);
```

Set the OutlineLevel property internally, so that there are four levels in the table of contents.

```
for i = 1:4
    % set internally the OutlineLevel property
    append(d,Heading(i,'My Chapter'));
    append(d,Paragraph('chapter content...'));
end
```

Use OutlineLevel to set the level of the Glossary paragraph to 1, so that the paragraph appears at the top level of the table of contents. Display the report.

```
para = append(d,Paragraph('Glossary'));
para.Style = {OutlineLevel(1)};

close(d);
rptview(d.OutputPath,d.Type);
```

### **See Also**

mlreportgen.dom.Paragraph | mlreportgen.dom.Heading

### **Topics**

“Automatically Number Document Content” on page 13-101

# mlreportgen.dom.Page class

**Package:** mlreportgen.dom

Create page number placeholder

## Description

Create a placeholder for a page number. This object applies only to Word and PDF output. For Word output, opening the Word document causes Word to replace this object with the current page number. For PDF output, the DOM API replaces this object with the current page number when writing the document.

## Construction

PageNum = Page() creates a current page number object.

## Output Arguments

**PageNum — Current page number**

mlreportgen.dom.Page object

Current page number, returned as an mlreportgen.dom.Page object.

## Properties

**Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**CustomAttributes — Custom attributes of this element**

array of mlreportgen.dom.CustomAttribute objects

Custom attributes of this element, specified as an array of mlreportgen.dom.CustomAttribute objects. Use custom attributes supported by the output format.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### **Style — Format specification**

array of format objects

Format objects that specify the format of a document element.

### **StyleName — Style to apply from style sheet**

character vector

Name of the style to apply from the style sheet, specified as a character vector.

## **Examples**

### **Insert Current Page Number**

This example uses `mlreportgen.dom.Page` to insert the current page number in the footer of a document.

```
import mlreportgen.dom.*;
d = Document('mydoc','pdf');
open(d);

% Create page footer
footer = PDFPageFooter('default');
d.CurrentPageLayout.PageFooters = footer;

% Define page number string and add to footer.
d.CurrentPageLayout.FirstPageNumber = 1;
t = Text('Page ');
pageinfo = Paragraph();
pageinfo.WhiteSpace = 'preserve';
pageinfo.HAlign = 'center';
append(pageinfo,t);
append(pageinfo,Page());
append(footer,pageinfo);

% Create several pages.
p = Paragraph('Hello World');
append(d,p);
p = Paragraph('Another page');
p.Style = {PageBreakBefore(true)};
append(d,p);
append(d,clone(p));
```

```
close(d);  
rptview(d.OutputPath);
```

## **See Also**

[mlreportgen.dom.NumPages](#) | [mlreportgen.dom.PageNumber](#) | [mlreportgen.dom.Text](#) | [mlreportgen.dom.Paragraph](#)

## **Topics**

“Add Complex Page Numbers in Microsoft Word” on page 13-155

“Report Formatting Approaches” on page 13-17

## **Introduced in R2016a**

## mlreportgen.dom.PageBorder class

**Package:** mlreportgen.dom

Page borders for Microsoft Word and PDF page layouts

### Description

Use an object of the mlreportgen.dom.PageBorder class to specify page borders for Microsoft Word and PDF reports.

The mlreportgen.dom.PageBorder class is a handle class.

### Class Attributes

ConstructOnLoad	true
HandleCompatible	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

pageBorder = mlreportgen.dom.PageBorder() creates a PageBorder object with default properties where all the segments are solid, black, and 0.5 point wide, the top and bottom segments are one point from the page margins, and the left and right segments are four points from the page margins.

pageBorder = mlreportgen.dom.PageBorder(style) specifies the default style for all border segments and sets the Style property to style.

pageBorder = mlreportgen.dom.PageBorder(style,color) also specifies the default color for all border segments and sets the Color property to color.

pageBorder = mlreportgen.dom.PageBorder(style,color,width) also specifies the default width for all border segments and sets the Width property to width.

pageBorder = mlreportgen.dom.PageBorder(style,color,width,margin) also specifies the default margin for all border segments and sets the Margin property to margin.

## Properties

### Style — Default style of page border segments

"solid" (default) | "single" | "dashed" | ...

Default style of the page border segments, specified as one of the values in the table.

<b>Border Style</b>	<b>Description</b>	<b>Supported Output Types</b>
"dashed"	Dashed line	Word, PDF
"dashdotstroked"	Line with alternating diagonal dashes and dot	Word
"dashsmallgap"	Dashed line with a small gap between dashes	Word
"dotted"	Dotted line	Word, PDF
"dotdash"	Line with alternating dots and dashes	Word
"dotdotdash"	Line with alternating double dots and a dash	Word
"double"	Double line	Word, PDF
"doublewave"	Double wavy line	Word
"inset"	3-D effect line	Word, PDF
"none"	No border	Word, PDF
"outset"	3-D effect line	Word, PDF
"single"	Single line	Word
"solid"	Single line	PDF
"thick"	Thick line	DOCX
"thickthinlargegap"	Dashed line with alternating thick and thin dashes with a large gap	DOCX
"thickthinmediumgap"	Dashed line with alternating thick and thin dashes with a medium gap	DOCX
"thickthinsmallgap"	Dashed line with alternating thick and thin dashes with a small gap	DOCX
"thinthicklargegap"	Dashed line with alternating thin and thick dashes with a large gap	DOCX
"thinthickmediumgap"	Dashed line with alternating thin and thick dashes with a medium gap	DOCX
"thinthicksmallgap"	Dashed line with alternating thin and thick dashes with a small gap	DOCX
"thinthickthinlargegap"	Dashed line with alternating thin and thick dashes with a large gap	DOCX
"thinthickthinmediumgap"	Dashed line with alternating thin and thick dashes with a medium gap	DOCX

Border Style	Description	Supported Output Types
"thinthickthinsmallgap"	Dashed line with alternating thin and thick dashes with a small gap	DOCX
"threedemboss"	Embossed effect line	DOCX
"threedengrave"	Engraved effect line	DOCX
"triple"	Triple line	DOCX
"wave"	Wavy line	DOCX

**Attributes:**

```

GetAccess                public
SetAccess                public
NonCopyable              true

```

**Color — Default color of page border segments**

"black" (default) | character vector | string scalar

Default color of the border segments, specified as a string scalar or character vector that contains the color name or equivalent hexadecimal RGB specification. See <https://www.w3.org/TR/2018/REC-css-color-3-20180619/>.

Example: "red"

Example: "#FF0000"

**Attributes:**

```

GetAccess                public
SetAccess                public
NonCopyable              true

```

**Width — Default width of page border segments**

"0.5pt" (default) | character vector | string scalar

Default width of the page border segments, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, "1pt" specifies one point. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Note** Whether Word honors the **Width** setting depends on the style specified by the **Style** property. For example, Word supports only 0.75-point and 1.5-point widths for the **wave** style. To see the widths that Word supports for a border style, in Word, on the **Design** tab, under **Page Background**, select **Page Borders**. Select a border under **Style**, then click the **Width** list to see the available widths.



Example: "1pt"

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Margin — Default margin of page border segments**

[ ] (default) | character vector | string scalar

Default margin of the page border segments, specified as a string scalar or character vector that consists of a number followed by an abbreviation for a unit of measurement. For example, "10pt" specifies ten points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

For PDF documents, the margin of a border segment is the space between the segment and the page margin. For Word documents, the margin of a border segment is the space between the segment and the page margin or between the segment and the edge of the page, depending on the value of the MeasureFrom property.

For Word documents, the Margin value must be between 0 points and 31 points, or the equivalent value using other units of measurement.

Example: "10pt"

**Attributes:**

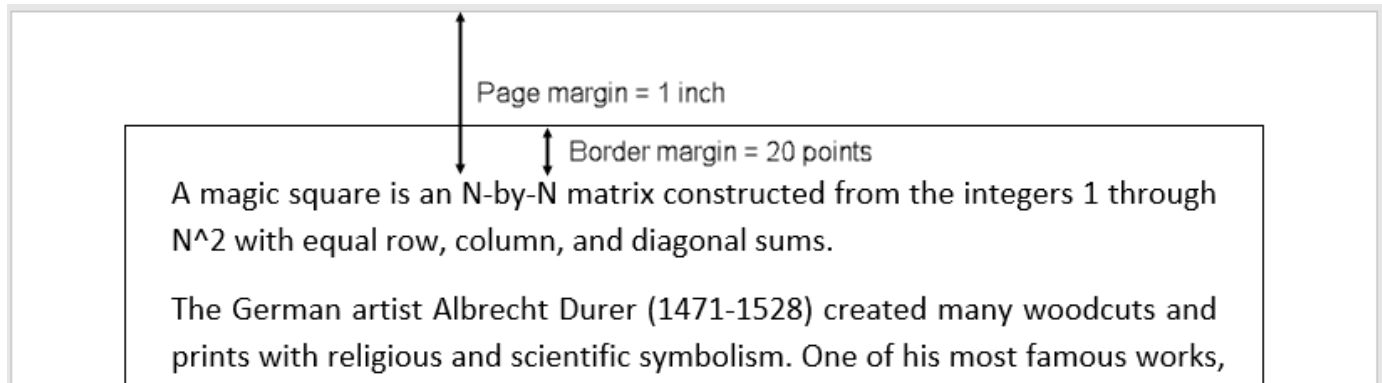
GetAccess	public
SetAccess	public
NonCopyable	true

**MeasureFrom — Whether border position is measured from page margin or edge of page**

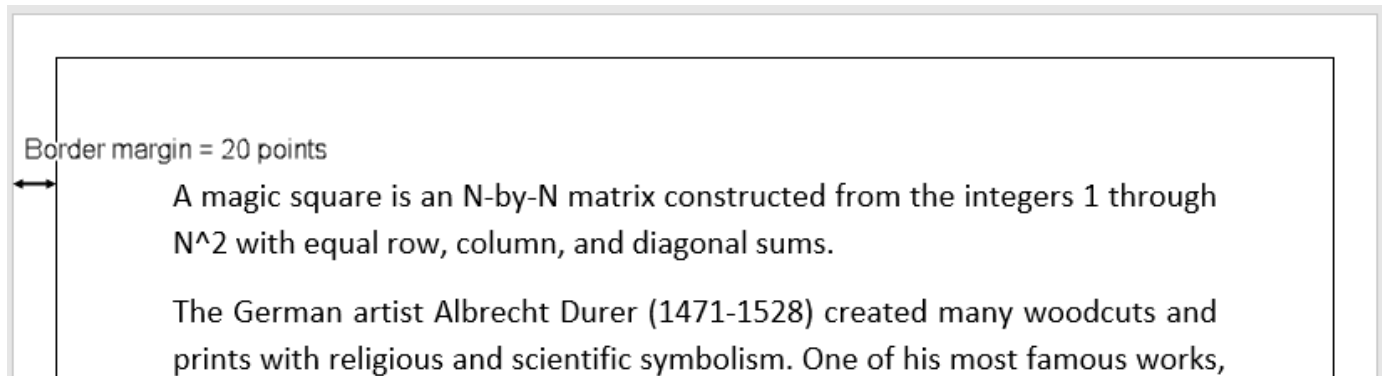
"text" (default) | "pageboundary"

Whether the border position is measured from the edge of the page or the edge of the page margin, which is the edge of the text, specified as "text" or "pageboundary". The "pageboundary" value applies only to Word documents. For a PDF document, the page border is always measured from the page margin.

If the value is "text", the Margin, TopMargin, LeftMargin, BottomMargin, and RightMargin properties specify the distance between a border segment and the page margin. For example, in this Word document, the page margin is one inch and the border is twenty points from the page margin.



If the value is "pageboundary", the value of the `Margin`, `TopMargin`, `LeftMargin`, `BottomMargin`, and `RightMargin` properties specify the distance between the border and the edge of the page. For example, the border is 20 points from the edge of the page in this Word document:



**Attributes:**

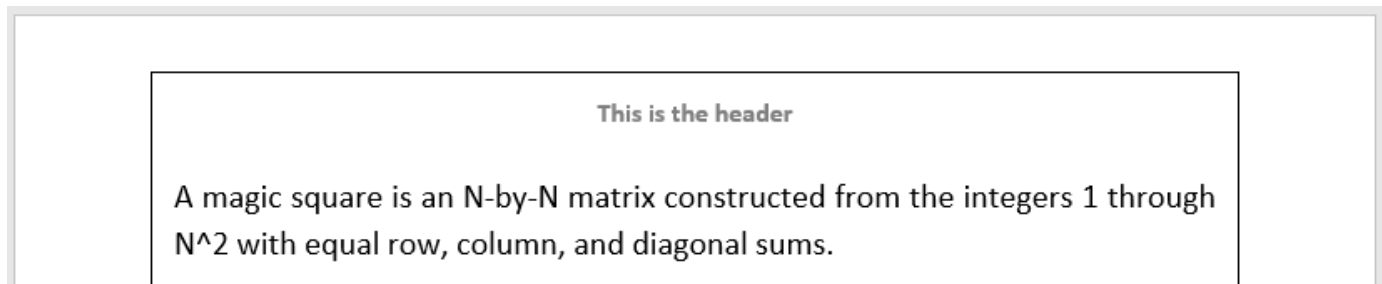
<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>
<code>NonCopyable</code>	<code>true</code>

**SurroundHeader — Whether page border surrounds header region**

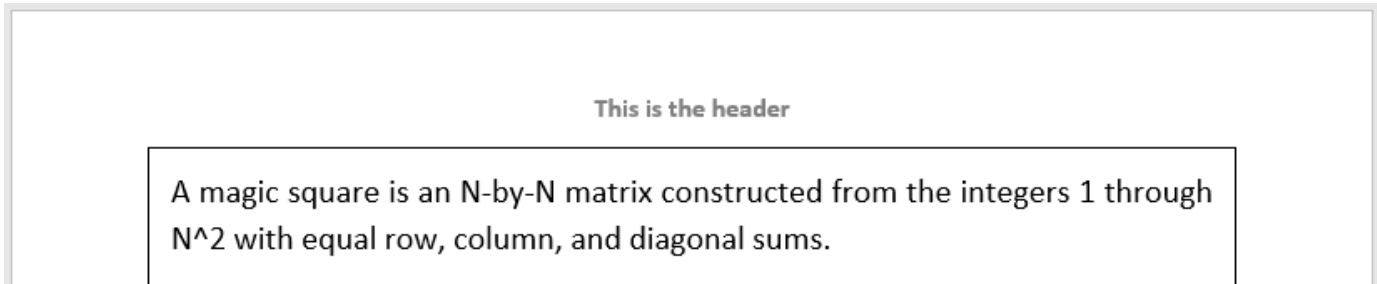
`true` (default) | `false`

Whether the page border surrounds the header region, specified as `true` or `false`. This property applies to PDF documents or to Word documents when the `MeasureFrom` property is set to "text".

If the value is `true`, the border surrounds the header. For example, the border surrounds the header in this Word document:



If the value is `false`, the border does not surround the header. For example, The border does not surround the header in this Word document:



**Note** For Word documents with multiple sections, a `false` value for the `SurroundHeader` property, or the equivalent setting in a template, applies to all sections of the document. If you specify that a page border does not surround the header for one section of the document, a page border does not surround the header for any section.

**Attributes:**

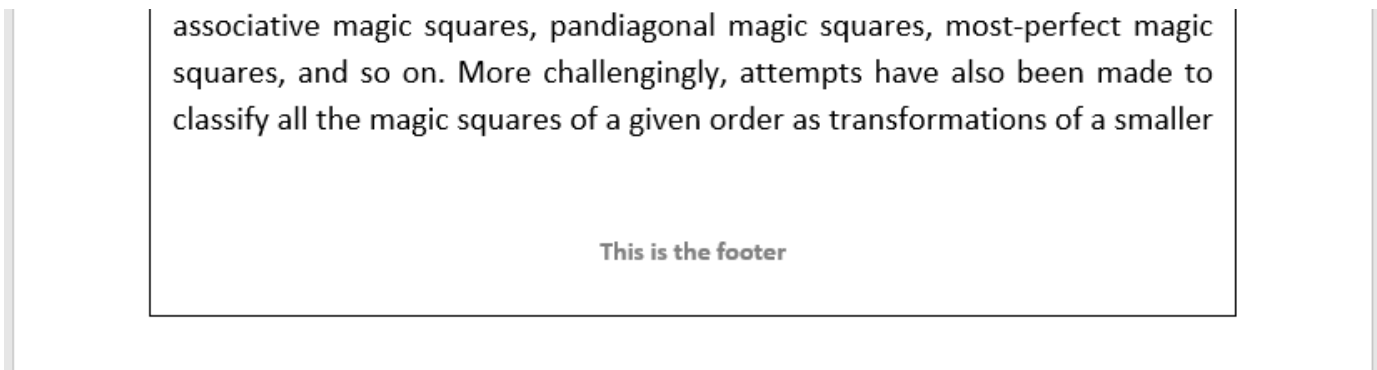
<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>
<code>NonCopyable</code>	<code>true</code>

**SurroundFooter — Whether page border surrounds footer region**

`true` (default) | `false`

Whether the page border surrounds the footer region, specified as `true` or `false`. This property applies to PDF documents or to Word documents when the `MeasureFrom` property is set to "text".

If the value is `true`, the border surrounds the footer. For example, the border surrounds the footer in this Word document:



If the value is `false`, the border does not surround the footer. For example, the border does not surround the footer in this Word document:

associative magic squares, pandiagonal magic squares, most-perfect magic squares, and so on. More challengingly, attempts have also been made to classify all the magic squares of a given order as transformations of a smaller

This is the footer

**Note** For Word documents with multiple sections, a `false` value for the `SurroundFooter` property, or the equivalent setting in a template, applies to all sections of the document. If you specify that a page border does not surround the footer for one section of the document, a page border does not surround the footer for any section.

**Attributes:**

GetAccess public  
 SetAccess public  
 NonCopyable true

**TopStyle — Style of top page border segment**

"solid" (default) | "single" | "dashed" | ...

Style of the top page border segment, specified as one of the values in the table.

Border Style	Description	Supported Output Types
"dashed"	Dashed line	Word, PDF
"dashdotstroked"	Line with alternating diagonal dashes and dot	Word
"dashsmallgap"	Dashed line with a small gap between dashes	Word
"dotted"	Dotted line	Word, PDF
"dotdash"	Line with alternating dots and dashes	Word
"dotdotdash"	Line with alternating double dots and a dash	Word
"double"	Double line	Word, PDF
"doublewave"	Double wavy line	Word
"inset"	3-D effect line	Word, PDF
"none"	No border	Word, PDF
"outset"	3-D effect line	Word, PDF
"single"	Single line	Word

Border Style	Description	Supported Output Types
"solid"	Single line	PDF
"thick"	Thick line	DOCX
"thickthinlargegap"	Dashed line with alternating thick and thin dashes with a large gap	DOCX
"thickthinmediumgap"	Dashed line with alternating thick and thin dashes with a medium gap	DOCX
"thickthinsmallgap"	Dashed line with alternating thick and thin dashes with a small gap	DOCX
"thinthicklargegap"	Dashed line with alternating thin and thick dashes with a large gap	DOCX
"thinthickmediumgap"	Dashed line with alternating thin and thick dashes with a medium gap	DOCX
"thinthicksmallgap"	Dashed line with alternating thin and thick dashes with a small gap	DOCX
"thinthickthinlargegap"	Dashed line with alternating thin and thick dashes with a large gap	DOCX
"thinthickthinmediumgap"	Dashed line with alternating thin and thick dashes with a medium gap	DOCX
"thinthickthinsmallgap"	Dashed line with alternating thin and thick dashes with a small gap	DOCX
"threedemboss"	Embossed effect line	DOCX
"threedengrave"	Engraved effect line	DOCX
"triple"	Triple line	DOCX
"wave"	Wavy line	DOCX

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**TopColor – Color of top border segment**

'black' (default) | character vector | string scalar

Color of the top border segment, specified as a string scalar or character vector that contains the color name or equivalent hexadecimal RGB specification. See <https://www.w3.org/TR/2018/REC-css-color-3-20180619/>.

Example: "red"

Example: "#FF0000"

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**TopWidth — Width of top border segment**

'0.5pt' (default) | character vector | string scalar

Width of the top border segment, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, "1pt" specifies one point. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

---

**Note** Whether Word honors the TopWidth setting depends on the style specified by the TopStyle property. For example, Word supports only 0.75-point and 1.5-point widths for the wave style. To see the widths that Word supports for a border style, in Word, on the **Design** tab, under **Page Background**, select **Page Borders**. Select a border under **Style**, then click the **Width** list to see the available widths.

---

Example: "1pt"

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**TopMargin — Margin of top border segment**

'1pt' (default) | character vector | string scalar

Margin of the top border segment, specified as a string scalar or character vector that consists of a number followed by an abbreviation for a unit of measurement. For example, "10pt" specifies ten points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas

- pt — points

For PDF documents, the margin of a border segment is the space between the segment and the page margin. For Word documents, the margin of a border segment is the space between the segment and the page margin or between the segment and the edge of the page, depending on the value of the `MeasureFrom` property.

For Word documents, the `TopMargin` value must be between 0 points and 31 points, or the equivalent value using other units of measurement.

Example: "10pt"

#### Attributes:

```
GetAccess                public
SetAccess                public
NonCopyable              true
```

#### LeftStyle — Style of left border segment

"solid" (default) | "single" | "dashed" | ...

Style of the left border segment, specified as one of the values in the table.

Border Style	Description	Supported Output Types
"dashed"	Dashed line	Word, PDF
"dashdotstroked"	Line with alternating diagonal dashes and dot	Word
"dashsmallgap"	Dashed line with a small gap between dashes	Word
"dotted"	Dotted line	Word, PDF
"dotdash"	Line with alternating dots and dashes	Word
"dotdotdash"	Line with alternating double dots and a dash	Word
"double"	Double line	Word, PDF
"doublewave"	Double wavy line	Word
"inset"	3-D effect line	Word, PDF
"none"	No border	Word, PDF
"outset"	3-D effect line	Word, PDF
"single"	Single line	Word
"solid"	Single line	PDF
"thick"	Thick line	DOCX
"thickthinlargegap"	Dashed line with alternating thick and thin dashes with a large gap	DOCX

Border Style	Description	Supported Output Types
"thickthinmediumgap"	Dashed line with alternating thick and thin dashes with a medium gap	DOCX
"thickthinsmallgap"	Dashed line with alternating thick and thin dashes with a small gap	DOCX
"thinthicklargegap"	Dashed line with alternating thin and thick dashes with a large gap	DOCX
"thinthickmediumgap"	Dashed line with alternating thin and thick dashes with a medium gap	DOCX
"thinthicksmallgap"	Dashed line with alternating thin and thick dashes with a small gap	DOCX
"thinthickthinlargegap"	Dashed line with alternating thin and thick dashes with a large gap	DOCX
"thinthickthinmediumgap"	Dashed line with alternating thin and thick dashes with a medium gap	DOCX
"thinthickthinsmallgap"	Dashed line with alternating thin and thick dashes with a small gap	DOCX
"threedemboss"	Embossed effect line	DOCX
"threedengrave"	Engraved effect line	DOCX
"triple"	Triple line	DOCX
"wave"	Wavy line	DOCX

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**LeftColor — Color of left border segment**

'black' (default) | character vector | string scalar

Color of the left border segment, specified as a string scalar or character vector that contains the color name or equivalent hexadecimal RGB specification. See <https://www.w3.org/TR/2018/REC-css-color-3-20180619/>.

Example: "red"

Example: "#FF0000"



**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**LeftWidth — Width of left border segment**

'0.5pt' (default) | character vector | string scalar

Width of the left border segment, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, "1pt" specifies one point. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

---

**Note** Whether Word honors the LeftWidth setting depends on the style specified by the LeftStyle property. For example, Word supports only 0.75-point and 1.5-point widths for the wave style. To see the widths that Word supports for a border style, in Word, on the **Design** tab, under **Page Background**, select **Page Borders**. Select a border under **Style**, then click the **Width** list to see the available widths.

---

Example: "1pt"

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**LeftMargin — Margin of left border segment**

'4pt' (default) | character vector | string scalar

Margin of the left border segment, specified as a string scalar or character vector that consists of a number followed by an abbreviation for a unit of measurement. For example, "10pt" specifies ten points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

For PDF documents, the margin of a border segment is the space between the segment and the page margin. For Word documents, the margin of a border segment is the space between the segment and

the page margin or between the segment and the edge of the page, depending on the value of the `MeasureFrom` property.

For Word documents, the `LeftMargin` value must be between 0 points and 31 points, or the equivalent value using other units of measurement.

Example: "10pt"

**Attributes:**

```
GetAccess                public
SetAccess                public
NonCopyable              true
```

**BottomStyle – Style of bottom border segment**

"solid" (default) | "single" | "dashed" | ...

Style of the bottom border segment, specified as one of the values in the table.

Border Style	Description	Supported Output Types
"dashed"	Dashed line	Word, PDF
"dashdotstroked"	Line with alternating diagonal dashes and dot	Word
"dashsmallgap"	Dashed line with a small gap between dashes	Word
"dotted"	Dotted line	Word, PDF
"dotdash"	Line with alternating dots and dashes	Word
"dotdotdash"	Line with alternating double dots and a dash	Word
"double"	Double line	Word, PDF
"doublewave"	Double wavy line	Word
"inset"	3-D effect line	Word, PDF
"none"	No border	Word, PDF
"outset"	3-D effect line	Word, PDF
"single"	Single line	Word
"solid"	Single line	PDF
"thick"	Thick line	DOCX
"thickthinlargegap"	Dashed line with alternating thick and thin dashes with a large gap	DOCX
"thickthinmediumgap"	Dashed line with alternating thick and thin dashes with a medium gap	DOCX
"thickthinsmallgap"	Dashed line with alternating thick and thin dashes with a small gap	DOCX

Border Style	Description	Supported Output Types
"thinthicklargegap"	Dashed line with alternating thin and thick dashes with a large gap	DOCX
"thinthickmediumgap"	Dashed line with alternating thin and thick dashes with a medium gap	DOCX
"thinthicksmallgap"	Dashed line with alternating thin and thick dashes with a small gap	DOCX
"thinthickthinlargegap"	Dashed line with alternating thin and thick dashes with a large gap	DOCX
"thinthickthinmediumgap"	Dashed line with alternating thin and thick dashes with a medium gap	DOCX
"thinthickthinsmallgap"	Dashed line with alternating thin and thick dashes with a small gap	DOCX
"threedemboss"	Embossed effect line	DOCX
"threedengrave"	Engraved effect line	DOCX
"triple"	Triple line	DOCX
"wave"	Wavy line	DOCX

**Attributes:**

```

GetAccess                public
SetAccess                public
NonCopyable              true

```

**BottomColor — Color of bottom border segment**

'black' (default) | character vector | string scalar

Color of the bottom border segment, specified as a string scalar or character vector that contains the color name or equivalent hexadecimal RGB specification. See <https://www.w3.org/TR/2018/REC-css-color-3-20180619/>.

Example: "red"

Example: "#FF0000"

**Attributes:**

```

GetAccess                public
SetAccess                public
NonCopyable              true

```

**BottomWidth — Width of bottom border segment**

'0.5pt' (default) | character vector | string scalar

Width of the bottom border segment, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, "1pt" specifies one point. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

---

**Note** Whether Word honors the `BottomWidth` setting depends on the style specified by the `BottomStyle` property. For example, Word supports only 0.75-point and 1.5-point widths for the wave style. To see the widths that Word supports for a border style, in Word, on the **Design** tab, under **Page Background**, select **Page Borders**. Select a border under **Style**, then click the **Width** list to see the available widths.

---

Example: "1pt"

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>
<code>NonCopyable</code>	<code>true</code>

**BottomMargin — Margin of bottom border segment**

'1pt' (default) | character vector | string scalar

Margin of the bottom border segment, specified as a string scalar or character vector that consists of a number followed by an abbreviation for a unit of measurement. For example, "10pt" specifies ten points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

For PDF documents, the margin of a border segment is the space between the segment and the page margin. For Word documents, the margin of a border segment is the space between the segment and the page margin or between the segment and the edge of the page, depending on the value of the `MeasureFrom` property.

For Word documents, the `BottomMargin` value must be between 0 points and 31 points, or the equivalent value using other units of measurement.

Example: "10pt"

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**RightStyle – Style of right border segment**

"solid" (default) | "single" | "dashed" | ...

Style of the right border segment, specified as one of the values in the table.

Border Style	Description	Supported Output Types
"dashed"	Dashed line	Word, PDF
"dashdotstroked"	Line with alternating diagonal dashes and dot	Word
"dashsmallgap"	Dashed line with a small gap between dashes	Word
"dotted"	Dotted line	Word, PDF
"dotdash"	Line with alternating dots and dashes	Word
"dotdotdash"	Line with alternating double dots and a dash	Word
"double"	Double line	Word, PDF
"doublewave"	Double wavy line	Word
"inset"	3-D effect line	Word, PDF
"none"	No border	Word, PDF
"outset"	3-D effect line	Word, PDF
"single"	Single line	Word
"solid"	Single line	PDF
"thick"	Thick line	DOCX
"thickthinlargegap"	Dashed line with alternating thick and thin dashes with a large gap	DOCX
"thickthinmediumgap"	Dashed line with alternating thick and thin dashes with a medium gap	DOCX
"thickthinsmallgap"	Dashed line with alternating thick and thin dashes with a small gap	DOCX
"thinthicklargegap"	Dashed line with alternating thin and thick dashes with a large gap	DOCX
"thinthickmediumgap"	Dashed line with alternating thin and thick dashes with a medium gap	DOCX

Border Style	Description	Supported Output Types
"thinthicksmallgap"	Dashed line with alternating thin and thick dashes with a small gap	DOCX
"thinthickthinlargegap"	Dashed line with alternating thin and thick dashes with a large gap	DOCX
"thinthickthinmediumgap"	Dashed line with alternating thin and thick dashes with a medium gap	DOCX
"thinthickthinsmallgap"	Dashed line with alternating thin and thick dashes with a small gap	DOCX
"threedemboss"	Embossed effect line	DOCX
"threedengrave"	Engraved effect line	DOCX
"triple"	Triple line	DOCX
"wave"	Wavy line	DOCX

**Attributes:**

```

GetAccess                public
SetAccess                public
NonCopyable              true

```

**RightColor — Color of right border segment**

'black' (default) | character vector | string scalar

Color of the right border segment, specified as a string scalar or character vector that contains the color name or equivalent hexadecimal RGB specification. See <https://www.w3.org/TR/2018/REC-css-color-3-20180619/>.

Example: "red"

Example: "#FF0000"

**Attributes:**

```

GetAccess                public
SetAccess                public
NonCopyable              true

```

**RightWidth — Width of right border segment**

'0.5pt' (default) | character vector | string scalar

Width of the right border segment, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, "1pt" specifies one point. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches

- mm — millimeters
- pc — picas
- pt — points

---

**Note** Whether Word honors the `RightWidth` setting depends on the style specified by the `RightStyle` property. For example, Word supports only 0.75-point and 1.5-point widths for the `wave` style. To see the widths that Word supports for a border style, in Word, on the **Design** tab, under **Page Background**, select **Page Borders**. Select a border under **Style**, then click the **Width** list to see the available widths.

---

Example: "1pt"

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>
<code>NonCopyable</code>	<code>true</code>

**RightMargin — Margin of right border segment**

'4pt' (default) | character vector | string scalar

Margin of the right border segment, specified as a string scalar or character vector that consists of a number followed by an abbreviation for a unit of measurement. For example, "10pt" specifies ten points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

For PDF documents, the margin of a border segment is the space between the segment and the page margin. For Word documents, the margin of a border segment is the space between the segment and the page margin or between the segment and the edge of the page, depending on the value of the `MeasureFrom` property.

For Word documents, the `RightMargin` value must be between 0 points and 31 points, or the equivalent value using other units of measurement.

Example: "10pt"

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>
<code>NonCopyable</code>	<code>true</code>

**Tag — Tag for this DOM API object**

character vector | string scalar

Tag for this DOM API object, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Id — ID for this DOM API object**

character vector | string scalar

ID for this DOM API object, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

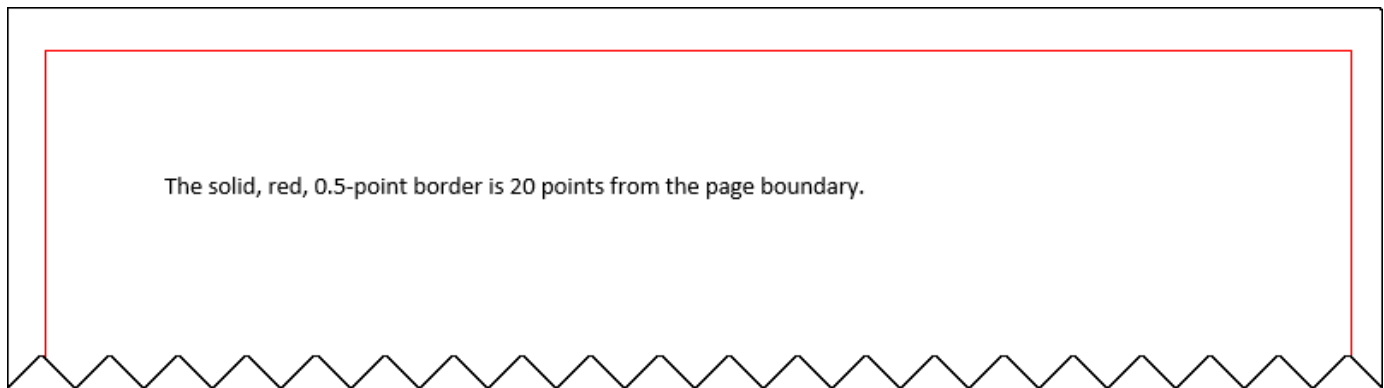
**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

## Examples

### Generate a Microsoft Word Document with Page Borders

This example generates a Microsoft Word document that has solid, red, 0.5-point borders positioned 20 points from the page boundary.



Import the DOM API package so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*;
```

Create a Word document.

```
d = Document("myDocument", "docx");
open(d);
```

Create a PageBorder object and specify the border style, color, and width.



```
pageBorder = PageBorder("solid", "red", "0.5pt");
```

Specify that the border position is relative to the page boundary and specify the margin between the page boundary and the border.

```
pageBorder.MeasureFrom = "pageboundary";
pageBorder.Margin = "20pt";
```

Set the PageBorder property of the layout associated with the document to the PageBorder object.

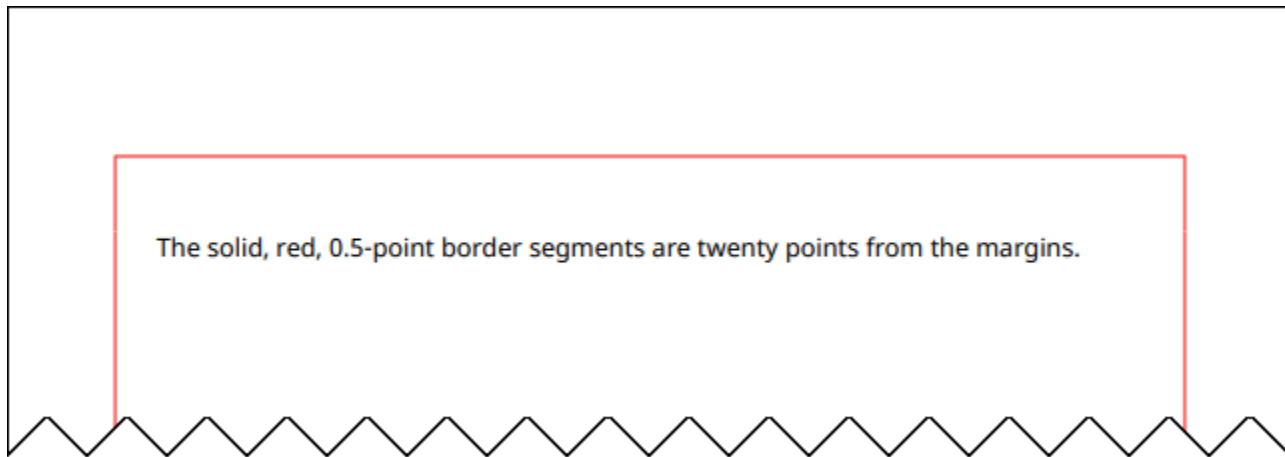
```
d.CurrentPageLayout.PageBorder = pageBorder;
```

Add text to the document. Close and view the document.

```
append(d, "The solid, red, 0.5-point border is 20 points from the page boundary.");
close(d);
rptview(d);
```

### Generate a PDF Document with Page Borders

This example generates a PDF document that has solid, red, 0.5-point borders positioned 20 points from the page margins.



Import the DOM API package so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*;
```

Create a PDF document.

```
d = Document("myDocument", "pdf");
open(d);
```

Create a PageBorder object and specify the style, color, width, and margin for all border segments. For PDF Documents, the margin of a border segment specifies the distance between the segment and the page margin.

```
pageBorder = PageBorder("solid", "red", "0.5pt", "20pt");
```

Set the PageBorder property of the layout associated with the document to the PageBorder object.

```
d.CurrentPageLayout.PageBorder = pageBorder;
```

Add text to the document. Close and view the document.

```
append(d, "The solid, red, 0.5-point border segments are twenty points from the margins.");
close(d);
rptview(d);
```

### Generate a Report API Report with Page Borders

This example uses the Report API to generate a Word or PDF report that has red page borders on the title page and blue page borders on the other sections of the report.

Import the Report API and DOM API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*;
import mlreportgen.dom.*;
```

Create a Word report and specify solid, blue, 0.5-point default borders for all pages of the report. For a PDF report, replace "docx" with "pdf".

```
rpt = Report("myreport", "docx");
rpt.Layout.PageBorder = PageBorder("solid", "blue", "0.5pt");
```

Create a title page and specify solid, red, 0.5-point borders for the title page. Append the title page to the report.

```
tp = TitlePage("Title", "My Report");
tp.Layout.PageBorder = PageBorder("solid", "red", "0.5pt");
append(rpt, tp);
```

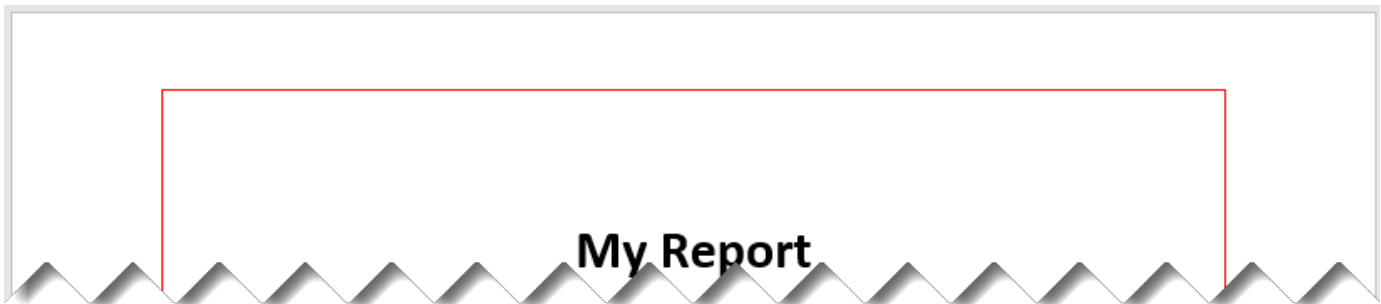
Create a chapter and append it to the report.

```
ch = Chapter("My Chapter");
append(rpt, ch);
```

Close and view the report.

```
close(rpt);
rptview(rpt);
```

In the generated report, the title page has red borders as specified by the title page reporter layout.



The chapter pages have the default blue borders because the chapter reporter layout did not specify page borders.



### **See Also**

`mlreportgen.dom.DOCXPageLayout` | `mlreportgen.dom.PDFPageLayout` |  
`mlreportgen.report.ReporterLayout` | `mlreportgen.report.ReportLayout`

### **Topics**

“Create Page Layout Sections” on page 13-144

“Create Page Footers and Headers” on page 13-148

**Introduced in R2021b**

## mlreportgen.dom.PageBreak class

**Package:** mlreportgen.dom mlreportgen.dom

Page break in a Word or PDF report

### Description

Use `mlreportgen.dom.PageBreak` to insert a page break anywhere in a Microsoft Word or PDF report.

---

**Tip** Use an `mlreportgen.dom.PageBreakBefore` object to force a page break before a specific paragraph. For example, use `PageBreakBefore` to force chapters to start on a new page.

---

The `mlreportgen.dom.PageBreak` class is a `handle` class.

### Class Attributes

<code>HandleCompatible</code>	<code>true</code>
<code>ConstructOnLoad</code>	<code>true</code>

## Creation

### Description

`break = mlreportgen.dom.PageBreak()` creates a page break object.

## Properties

### Children — Children of this DOM API object

array of DOM API objects

Children of this DOM API object, specified as an array of DOM API objects.

### Attributes:

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>private</code>
<code>NonCopyable</code>	<code>true</code>

### CustomAttributes — Custom attributes of this element

array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. Use custom attributes supported by the output format.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Id — ID for this DOM API object**

character vector | string scalar

ID for this DOM API object, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Parent — Parent of this DOM API object**

DOM API object

Parent of this DOM API object, specified as a DOM API object.

**Attributes:**

GetAccess	public
SetAccess	private
NonCopyable	true

**Style — Text formatting**

array of format objects

Format objects that specify the format of a document element.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**StyleName — Style to apply from style sheet**

character vector | string scalar

Name of the style to apply from the style sheet, specified as a character vector or string scalar.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Tag — Tag for this DOM API object**

character vector | string scalar

Tag for this DOM API object, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the

object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Attributes:**

```
GetAccess           public
SetAccess           public
NonCopyable         true
```

**Methods**

**Public Methods**

Method	Purpose
clone	Copy this object. Use the clone method for a PageBreak object in the same way that you use the clone method for a Paragraph object.

**Examples**

**Insert a Page Break**

This example shows how to insert a page break in a PDF report by using a PageBreak object.

Import the DOM API package so that you do not have to use long, fully qualified names.

```
import mlreportgen.dom.*;
```

Create a Document object and open it.

```
d = Document("mydoc", "pdf");
open(d);
```

Create the first page consisting of a heading and some paragraphs.

```
h = Heading1("My First Heading");
append(d,h);
p = Paragraph("Here are some paragraphs.");
append(d,p);
for i = 1:5
append(d,clone(p));
end
```

Insert a page break by using a PageBreak object.

```
br = PageBreak();
append(d,br);
```

In the newly created second page, append some paragraphs.

```
p2 = Paragraph("Here are some paragraphs after the forced page break.");
append(d,p2);
for i = 1:5
append(d,clone(p2));
end
```

Close and view the document.

```
close(d);  
rptview(d);
```

### **See Also**

mlreportgen.dom.Paragraph | mlreportgen.dom.PageBreakBefore |  
mlreportgen.dom.KeepWithNext | mlreportgen.dom.KeepLinesTogether |  
mlreportgen.dom.WidowOrphanControl

### **Topics**

“Report Formatting Approaches” on page 13-17

### **Introduced in R2016a**

## mlreportgen.dom.PageBreakBefore class

**Package:** mlreportgen.dom

Start paragraph on new page

### Description

Specifies to always start the associated paragraph on a new page. This class applies to Microsoft Word and PDF reports.

The mlreportgen.dom.PageBreakBefore class is a handle class.

## Creation

### Description

pageBreakBefore = PageBreakBefore() always starts the paragraph on a new page.

pageBreakBefore = PageBreakBefore(onOff) always starts paragraph on a new page if onOff is true.

### Input Arguments

#### onOff — Option to start paragraph on new page

true (default) | false

Option to start paragraph on new page, specified as one of these values:

- true or 1 — Starts a paragraph on a new page.
- false or 0 — Allows a paragraph to start on the current page.

Data Types: logical

## Properties

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.



**Value — Option to start paragraph on new page**

true (default) | false

Option to start paragraph on new page, specified as one of these values:

- true or 1 — Starts a paragraph on a new page.
- false or 0 — Allows a paragraph to start on the current page.

Data Types: logical

**Examples****Create a Page Break Before a Heading**

This example shows how to apply the PageBreakBefore property to a heading paragraph. The example uses two approaches for applying properties. The first creates a PageBreakBefore object whose value is explicitly true. You can then assign that format object to the heading's Style property. The second approach sets the property on the heading object without explicitly creating a PageBreakBefore object.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'docx');
open(d);

% Create first page text
t = Heading(1, 'Document Title', 'Title');
h = Heading(2, 'My Head', 'Heading1');
p = Paragraph('Hello World');

append(d,t);
append(d,h);
append(d,p);

% Create a heading paragraph h1
% Create a PageBreakBefore object and set it as a Style property on h1
h1 = Heading(2, 'My Second Head', 'Heading1');
br = {PageBreakBefore(true)};
h1.Style = br;
p1 = Paragraph('Another page');

% Create a heading paragraph h2
% Set the h2 Style property to use PageBreakBefore set to true
h2 = Heading(2, 'My Third Head', 'Heading1');
h2.Style = {PageBreakBefore()};
p2 = Paragraph('My third page');

append(d,h1);
append(d,p1);
append(d,h2);
append(d,p2);
```

```
close(d);  
rptview(d.OutputPath);
```

### **See Also**

`m1reportgen.dom.Paragraph` | `m1reportgen.dom.PageBreak`

### **Topics**

“Report Formatting Approaches” on page 13-17

**Introduced in R2014b**

# mlreportgen.dom.PageMargins class

**Package:** mlreportgen.dom

Page margins for Microsoft Word and PDF page layout

## Description

Use an object of the `mlreportgen.dom.PageMargins` class to specify the size of the page margins of a section of a Microsoft Word or PDF report.

Creation of page headers and footers differs between Word and PDF documents. For example, PDF page headers and footers are fixed in size. Word headers and footers expand to fit the content. Therefore, the appearance of top and bottom page margins using the same values can differ based on the output type.

In addition, top and bottom margins are handled differently for different output types. These differences can result in different page breaks even with the same margin settings.

- For PDF, the total height of the top margin equals the value of the `Top` property of this object plus the height of the `Header` property. The body text starts below the header. For Word documents, the top margin and header expand to prevent overlapping of the header and the body text.
- Similarly, the total height of the PDF bottom margin equals the value of the `Bottom` property of this object plus the height of the `Footer` property. The body text ends above the footer. For Word documents, the footer expands to prevent overlapping of the body text.

The `mlreportgen.dom.PageMargins` class is a `handle` class.

## Class Attributes

<code>HandleCompatible</code>	<code>true</code>
<code>ConstructOnLoad</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`pageMarginsObj = mlreportgen.dom.PageMargins()` creates a `PageMargins` object with properties on page 12-363 that specify 1-inch top, bottom, left, and right margins, a 0.5-inch header and footer, and a 0-pixel gutter.

## Properties

### Top — Top margin size

'1in' (default) | character vector | string scalar

Top margin size, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '1in' specifies 1 inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Bottom — Bottom margin size**

'1in' (default) | character vector | string scalar

Bottom margin size, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '1in' specifies 1 inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Left — Left margin size**

'1in' (default) | character vector | string scalar

Left margin size, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '1in' specifies 1 inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Right — Right margin size**

'1in' (default) | character vector | string scalar

Right margin size, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '1in' specifies 1 inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters

- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**Header — Header size**

'0.5in' (default) | character vector | string scalar

Header size, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '0.5in' specifies 0.5 inches. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**Footer — Footer size**

'0.5in' (default) | character vector | string scalar

Footer size, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '0.5in' specifies 0.5 inches. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**Gutter — Gutter size**

'0px' (default) | character vector | string scalar

Gutter size, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '0px' specifies 0 pixels. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Examples

### Specify Page Margins in a DOM Document

In the `PageMargins` object associated with the document layout object, specify 0.5-inch margins.

```
import mlreportgen.dom.*;
d = Document('myreport', 'docx');
open(d);

pageLayoutObj = d.CurrentPageLayout;
pageLayoutObj.PageMargins.Left = '0.5in';
pageLayoutObj.PageMargins.Right = '0.5in';
append(d, 'Left and right margins are 0.5 inches.');
```

```
close(d);
rptview(d);
```

### Specify Page Margins in a Report API Report

Use an `mlreportgen.dom.PageMargins` object to specify 0.5-inch margins. Assign the object to the `PageMargins` property of the report layout object.

```
import mlreportgen.report.*;
import mlreportgen.dom.*;

rpt = Report('myreport', 'docx');
open(rpt);

pageMarginsObj = PageMargins();
pageMarginsObj.Left = '0.5in';
pageMarginsObj.Right = '0.5in';
rpt.Layout.PageMargins = pageMarginsObj;

append(rpt, 'Left and right margins are 0.5 inches.');
```

```
close(rpt);
rptview(rpt);
```

## **See Also**

mlreportgen.dom.DOCXPageLayout | mlreportgen.dom.PDFPageLayout |  
mlreportgen.dom.DOCXPageFooter | mlreportgen.dom.PDFPageFooter |  
mlreportgen.dom.DOCXPageHeader | mlreportgen.dom.PDFPageHeader |  
mlreportgen.report.ReportLayout | mlreportgen.report.ReporterLayout

## **Topics**

“Report Formatting Approaches” on page 13-17

“Customize the Page Size and Margins of a Report Programmatically” on page 17-154

## **Introduced in R2016a**

## mlreportgen.dom.PageNumber class

**Package:** mlreportgen.dom

Format page numbers

### Description

Specifies the initial value of a page number in a Word page layout and the type of number, such as roman numerals. Set this property as a style on a `DOCXPageLayout` or a `PDFPageLayout` object. Insert the page number in a page layout object using `mlreportgen.dom.Page` to see the effect of this format.

This object enables you to create compound page numbers in the form [Chapter #]-[Page#] and specify the separator. For an example, see “Add Complex Page Numbers in Microsoft Word” on page 13-155.

### Construction

`pgnum = PageNumber()` specifies a numeric page number whose value continues from the previous page.

`pgnum = PageNumber(number)` sets the initial value to the specified number.

`pgnum = PageNumber(number, format)` uses the specified type of number, such as roman numerals.

### Input Arguments

#### **number — Number of first page in this layout**

positive integer

Number of first page in this layout, specified as a positive integer.

#### **format — Type of page numbering to use**

character vector

Type of page numbering to use, specified as one of these values.

Value	Meaning	Applies To	
		DOCX	PDF
'a'	Lowercase alphabetic	✓	✓
'A'	Uppercase alphabetic	✓	✓
'i'	Lowercase Roman numerals	✓	✓
'I'	Uppercase Roman numerals	✓	✓



Value	Meaning	Applies To	
		DOCX	PDF
'n'. 'N' , '1', 'decimal'	Arabic numerals	✓	✓
'numberInDash'	Number with dashes on either side	✓	
'hebrew1'	Hebrew numerals	✓	
'hebrew2'	Hebrew alphabetic	✓	
'arabicAlpha'	Arabic alphabetic	✓	
'arabicAbjad'	Arabic abjad numerals	✓	
'thaiLetters'	Thai letters	✓	
'thaiNumbers'	Thai numerals	✓	
'thaiCounting'	Thai counting system	✓	

## Output Arguments

### pgNumber — Page number format

mlreportgen.dom.PageNumber object

Page number format, returned as an mlreportgen.dom.PageNumber object.

## Properties

### ChapterSeparator — Character to separate chapter number from page number

'colon' | ';' | 'hyphen' | '-' | 'emdash' | 'endash' | 'period' | '.'

Character to use to separate the chapter number from the page number, specified as one of these values:

- 'colon' or ';' — Colon.
- 'hyphen' or '-' — Hyphen.
- 'emdash' — Em dash (—).
- 'endash' — En dash (-).
- 'period' or '.' — Period.

### ChapterStartStyle — Level of Heading style that chapters use

positive integer character vector

Level of Heading style that chapters use, specified as a positive integer character vector.

### Format — Type of page numbering to use

character vector | string scalar

Type of page numbering to use, specified as one of the character vectors or string scalars in the table.

Value	Meaning	Applies To	
		DOCX	PDF
'a'	Lowercase alphabetic	✓	✓
'A'	Uppercase alphabetic	✓	✓
'i'	Lowercase Roman numerals	✓	✓
'I'	Uppercase Roman numerals	✓	✓
'n'. 'N', '1', 'decimal'	Arabic numerals	✓	✓
'numberInDash'	Number with dashes on either side	✓	
'hebrew1'	Hebrew numerals	✓	
'hebrew2'	Hebrew alphabetic	✓	
'arabicAlpha'	Arabic alphabetic	✓	
'arabicAbjad'	Arabic abjad numerals	✓	
'thaiLetters'	Thai letters	✓	
'thaiNumbers'	Thai numerals	✓	
'thaiCounting'	Thai counting system	✓	

### Id – ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### InitialValue – Value of first page number in this layout

positive integer

Value of first page number in this layout, specified as a positive integer.

### Tag – Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Examples

### Style Numbers Using PageNumber

This example shows the syntax for using `PageNumber`, which you set using `PageNumber` as a style on the current page layout. For the numbering to take effect, you need to insert a page number into a page footer or header using `Page`, and you need to use a multilevel list style in the Word template. For a complete example, see “Add Complex Page Numbers in Microsoft Word” on page 13-155

```
import mlreportgen.dom.*;
d = Document('mypages', 'docx');

open(d);
layout = d.CurrentPageLayout;

% Start on page 7 and use roman numerals
pagenumber = PageNumber(7, 'I');

% Add page number object to page layout styles
layout.Style = [layout.Style {pagenumber}];

% Create the footer and add a page number to it
myfooter = DOCXPageFooter();
para = Paragraph();
para.HAlign = 'center';
append(para, Page());

% Add the page number to the footer
append(myfooter, para);
layout.PageFooters = myfooter;

% Add content
append(d, 'Hello World');

close(d);
rptview(d);
```

### See Also

[mlreportgen.dom.DOCXPageLayout](#) | [mlreportgen.dom.PDFPageLayout](#) |  
[mlreportgen.dom.DOCXPageFooter](#) | [mlreportgen.dom.PDFPageFooter](#) |  
[mlreportgen.dom.Page](#)

### Topics

“Add Complex Page Numbers in Microsoft Word” on page 13-155

## mlreportgen.dom.PageRawFormat class

**Package:** mlreportgen.dom

XML markup for array of Microsoft Word formats

### Description

XML markup for an array of Microsoft Word formats.

### Construction

RawFormatObj = PageRawFormat() creates an empty array of raw formats.

### Output Arguments

#### RawFormatObj — XML markup for Word formats

mlreportgen.dom.PageRawFormat object

XML markup for Word formats, represented by an mlreportgen.dom.PageRawFormat object.

### Properties

#### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

#### Markup — Word XML markup

cell array of character vectors

Specify a cell array of character vectors. Each character vector contains Word XML markup for a Word format.

For information about XML markup for Word formats, see <https://www.ecma-international.org/publications-and-standards/standards/ecma-376/>.

#### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Examples

## Turn on Line Numbering Based on Default DOM Template

In this example, the `RawFormats` property of the `CurrentPageLayout` object is initialized with the markup for properties specified by the default template. This code appends the line numbering property to the existing properties.

```
import mlreportgen.dom.*;
d = Document('myreport', 'docx');
open(d);

s = d.CurrentPageLayout;
s.RawFormats = [s.RawFormats ...
{'<w:lnNumType w:countBy="1" w:start="0" w:restart="newSection"/>'}];
p = Paragraph('This document has line numbers');
append(d, 'This document has line numbers');
append(d, clone(p));

close(d);
rptview(d.OutputPath);
```

## See Also

`mlreportgen.dom.DOCXPageLayout`

## Topics

“Report Formatting Approaches” on page 13-17

## Introduced in R2016a

## mlreportgen.dom.PageRef class

**Package:** mlreportgen.dom

Create placeholder for reference to page number of link target

### Description

Create a placeholder for a reference to the page number of a link target. This object applies only to Word and PDF output. For Word output, opening a document causes Word to replace this object with the page number of the link target that this object specifies. For PDF output, the DOM API replaces this object with the page number of the link target that it specifies.

---

**Tip** Use this object to generate page references, such as "See page 15 for more information."

---

### Construction

`pageRef = PageRef(target)` creates a page reference object that refers to the specified `LinkTarget` object. Generating the output replaces this `PageRef` object with the number of the page that contains the specified target.

### Input Arguments

**target** — Name of link target to reference

character vector

Name of the link target to reference, specified as a character vector. Specify the target using a `LinkTarget` object.

### Output Arguments

**pageRef** — Name of link target to reference

mlreportgen.dom.PageRef object

Name of link target to reference, returned as an `mlreportgen.dom.PageRef` object. This `PageRef` object is replaced with the number of the page containing the specified `LinkTarget` object.

### Properties

**Children** — Children of document element

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**CustomAttributes** — Custom attributes of this element

array of mlreportgen.dom.CustomAttribute objects

Custom attributes of this element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. Use custom attributes supported by the output format.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Style — Format specification**

array of format objects

Format objects that specify the format of a document element.

**StyleName — Style to apply from style sheet**

character vector

Name of the style to apply from the style sheet, specified as a character vector.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Target — Name of target of this page reference**

character vector

Name of the target of this page reference, specified as a character vector.

**Methods**

Method	Purpose
clone	Copy this object.

**Examples****Insert a Page Number Reference**

This example inserts a page number reference to a target on another page. Add a target mytarget using LinkTarget. Use PageRef to refer to the page that contains the target mytarget.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'pdf');
open(d);
```

```
% Create page footer and add page number to it
```

```
footer = PDFPageFooter('default');
d.CurrentPageLayout.PageFooters = footer;
d.CurrentPageLayout.FirstPageNumber = 1;
pageno = Paragraph();
pageno.HAlign = 'center';
append(pageno,Page());
append(footer,pageno);

% Add target to heading object and append heading and para text to document
h = Heading1(LinkTarget('mytarget'));
append(h,'Head Whose Page to Reference');
p = Paragraph('Here is some paragraph text. ');
append(d,h);
append(d,p);

% Add another page and insert page reference to target
p1 = Paragraph('The following paragraph contains the page reference. ');
p1.Style = {PageBreakBefore(true)};
p2 = Paragraph('See Page ');
p2.WhiteSpace = 'preserve';
ref = PageRef('mytarget');
append(p2,ref);
append(p2,' ');
append(d,p1);
append(d,p2);

close(d);
rptview(d.OutputPath);
```

## See Also

[mlreportgen.dom.Text](#) | [mlreportgen.dom.Paragraph](#) | [mlreportgen.dom.Page](#) | [mlreportgen.dom.LinkTarget](#) | [mlreportgen.dom.XRef](#) | [mlreportgen.dom.StyleRef](#)

## Topics

“Create Links” on page 13-82

“Report Formatting Approaches” on page 13-17

## Introduced in R2016a



# mlreportgen.dom.PageSize class

**Package:** mlreportgen.dom

Size and orientation of pages in Microsoft Word and PDF reports

## Description

Use an object of the mlreportgen.dom.PageSize class to specify the height, width, and orientation of pages in a Microsoft Word or PDF report.

The mlreportgen.dom.PageSize class is a handle class.

### Class Attributes

ConstructOnLoad	true
HandleCompatible	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

PageSizeObj = mlreportgen.dom.PageSize() creates a PageSize object with properties on page 12-377 that specify a height of 11 inches, a width of 8.5 inches, and a portrait orientation.

PageSizeObj = mlreportgen.dom.PageSize(height,width) creates a page size object that has the specified height and width and a portrait orientation.

PageSizeObj = mlreportgen.dom.PageSize(height,width,orientation) creates a page size object that has the specified height, width, and orientation.

## Properties

### Height — Height of pages

'11in' (default) | character vector | string scalar

Height of pages, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '11in' specifies 11 inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Width — Width of pages**`'8.5in'` (default) | character vector | string scalar

Width of pages, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, `'8.5in'` specifies 8.5 inches. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**Orientation — Orientation of pages**`'portrait'` (default) | `'landscape'`

Orientation of pages, specified as one of these character vectors or string scalars:

- `'portrait'` for vertical orientation
- `'landscape'` for horizontal orientation

The `Height` and `Width` properties determine the page orientation, regardless of the value of the `Orientation` property. However, it is a best practice to set the `Orientation` property to a value that is consistent with the page dimensions. If the height is greater than the width, set the `Orientation` to `'portrait'`. If the width is greater than the height, set the `Orientation` to `'landscape'`.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

## Examples

**Specify Page Size and Orientation in a DOM Document**

Use a `PageSize` object to specify the page size of a document. In the `PageSize` object, set the `Orientation` property to a value that is consistent with the `Height` and `Width` properties.

By default, a document has pages with an 11-inch height, 8.5-inch width, and portrait orientation. You can override the default page size by setting the `Height` and `Width` properties of the `PageSize` object used by the document layout object. For example, specify that the `Height` is "10in", the `Width` is "8in", and the `Orientation` is "portrait".

```
import mlreportgen.dom.*;
d = Document("myreport1","docx");
open(d);

pageLayoutObj = d.CurrentPageLayout;
pageLayoutObj.PageSize.Height = "10in";
pageLayoutObj.PageSize.Width = "8in";
pageLayoutObj.PageSize.Orientation = "portrait";
append(d,"This document has portrait pages");

close(d);
rptview(d);
```

Specifying a height that is less than the width makes the orientation landscape, regardless of the value of the `Orientation` property. This example specifies a height of 8 inches and a width of 10 inches. The example sets the `Orientation` property to "landscape" to be consistent with the page size.

```
import mlreportgen.dom.*;
d = Document("myreport2","docx");
open(d);

pageLayoutObj = d.CurrentPageLayout;
pageLayoutObj.PageSize.Height = "8in";
pageLayoutObj.PageSize.Width = "10in";
pageLayoutObj.PageSize.Orientation = "landscape";
append(d,"This document has landscape pages");

close(d);
rptview(d);
```

Instead of changing the page orientation by changing the height and width, you can use the `rotate` method of the layout object. The method switches the `Height` and `Width` property values and changes the `Orientation` property from portrait to landscape or landscape to portrait.

```
import mlreportgen.dom.*;
d = Document("myreport3","docx");
open(d);

pageLayoutObj = d.CurrentPageLayout;
pageLayoutObj.PageSize.Height = "10in";
pageLayoutObj.PageSize.Width = "8in";
pageLayoutObj.PageSize.Orientation = "portrait";
rotate(pageLayoutObj);
append(d,"This document has landscape pages");

close(d);
rptview(d);
```

### Specify Page Size and Orientation in a Report API Report

Use a `PageSize` object to specify the page size of a report or section of a report. In the `PageSize` object, set the `Orientation` property to a value that is consistent with the `Height` and `Width` properties.

By default, a report has pages with an 11-inch height and 8.5-inch width. To override the default page size for a report, create a `PageSize` object and assign it to the report layout object. To override the page size for a report section, create a `PageSize` object and assign it to the layout object used by the section reporter object. This example specifies a 10-inch height and 8-inch width for the report. The title page overrides the report page size and the rest of the report uses the page size of the report.

```
import mlreportgen.report.*;
import mlreportgen.dom.*;

rpt = Report("myreport1", "docx");
open(rpt);

layoutObj = rpt.Layout;
layoutObj.PageSize = PageSize("10in", "8in", "portrait");

tp = TitlePage("Title", "Title Page");
tlayoutObj = tp.Layout;
tlayoutObj.PageSize = PageSize("11in", "9in", "portrait");
append(rpt, tp);

toc = TableOfContents;
append(rpt, toc);

ch = Chapter("My Chapter");
para = Paragraph("This chapter uses the report page size");

close(rpt);
rptview(rpt);
```

The default page size of a report has a portrait orientation. To change the orientation of all pages of a report, use the `Landscape` property of the report layout object. This example specifies landscape orientation for all pages of the report.

```
import mlreportgen.report.*;
import mlreportgen.dom.*;

rpt = Report("myreport2", "docx");
open(rpt);

layoutObj = rpt.Layout;
layoutObj.Landscape = true;

tp = TitlePage("Title", "Title Page", "SubTitle", "With Page Size and Orientation of Report");
append(rpt, tp);

close(rpt);
rptview(rpt);
```

To change the orientation of the pages in a section with respect to the orientation of the report pages, use the `Landscape` property of the section layout object. In this example, the report has the default

dimensions, 11-inch height and 8.5-inch width, which is portrait orientation. The example changes the orientation of the title page to landscape.

```
import mlreportgen.report.*;
import mlreportgen.dom.*;

rpt = Report("myreport3", "docx");
open(rpt);
tp = TitlePage("Title", "Title Page", "SubTitle", "With Landscape Orientation");
tlayoutObj = tp.Layout;
tlayoutObj.Landscape = true;
append(rpt, tp);

close(rpt);
rptview(rpt);
```

## See Also

[mlreportgen.dom.DOCXPageLayout](#) | [mlreportgen.dom.PDFPageLayout](#) |  
[mlreportgen.dom.PageMargins](#) | [mlreportgen.report.ReportLayout](#) |  
[mlreportgen.report.ReporterLayout](#)

## Topics

“Report Formatting Approaches” on page 13-17

“Customize the Page Size and Margins of a Report Programmatically” on page 17-154

## Introduced in R2016a

## mlreportgen.dom.Paragraph class

**Package:** mlreportgen.dom

Formatted block of text

### Description

Use an mlreportgen.dom.Paragraph object to define a paragraph. You can append document elements, such as an image, to a paragraph.

The mlreportgen.dom.Paragraph class is a handle class.

### Class Attributes

ConstructOnLoad	true
HandleCompatible	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

paragraphObj = mlreportgen.dom.Paragraph() creates an empty paragraph.

paragraphObj = mlreportgen.dom.Paragraph(text) creates a paragraph that contains the specified text.

paragraphObj = mlreportgen.dom.Paragraph(text, styleName) creates a paragraph that has the specified style. The style specified by the styleName property must be defined in the template used for the document element to which this paragraph is appended.

paragraphObj = mlreportgen.dom.Paragraph(docElementObj) creates a paragraph that contains the document element specified by docElementObj.

### Input Arguments

#### text — Paragraph text

character vector | string scalar

Paragraph text, specified as a character vector or string scalar.

#### styleName — Style for paragraph

character vector | string scalar

Style for this paragraph, specified as a character vector or string scalar. The style must be defined in the template used to create the document to which this paragraph is appended.

#### docElementObj — Document element to include in paragraph

DOM object

Document element (DOM object) to include in the paragraph. You can specify these DOM objects:

- `mlreportgen.dom.ExternalLink`
- `mlreportgen.dom.Image`
- `mlreportgen.dom.InternalLink`
- `mlreportgen.dom.LinkTarget`
- `mlreportgen.dom.Text`

## Properties

### **OutlineLevel — Outline level of paragraph**

`[]` (default) | integer

Outline level of the paragraph, specified as an integer. Setting the `OutlineLevel` property causes this paragraph to be included in automatically generated outlines, such as a table of contents. The value specifies the level of the paragraph in the outline. For example, to make a paragraph appear at the top level in an outline, set the `OutlineLevel` property to 1.

Setting the `OutlineLevel` property adds a corresponding `mlreportgen.dom.OutlineLevel` format object to the `Style` property. Setting the `OutlineLevel` property to an empty value removes the object.

### **Bold — Whether to use bold for text**

`[]` (default) | `true` or 1 | `false` or 0

Whether to use bold for text, specified as a numeric or logical 1 (`true`) or 0 (`false`). To make text bold, set `Bold` to `true` or 1.

Setting the `Bold` property adds a corresponding `mlreportgen.dom.Bold` format object to the `Style` property. Setting the `Bold` property to an empty value removes the object.

### **Italic — Whether to use italic for text**

`[]` (default) | `true` or 1 | `false` or 0

Whether to use italic for text, specified as a numeric or logical 1 (`true`) or 0 (`false`). To make text italic, set `Italic` to `true` or 1.

Setting the `Italic` property adds a corresponding `mlreportgen.dom.Italic` format object to the `Style` property. Setting the `Italic` property to an empty value removes the object.

### **Color — Text color**

`[]` (default) | character vector | string scalar

Text color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form `#RRGGBB`.

Setting the `Color` property adds a corresponding `mlreportgen.dom.Color` format object to the `Style` property. Setting the `Color` property to an empty value removes the object.

Example: `'blue'`

Example: `'#0000ff'`

**BackgroundColor – Background color**

[] (default) | character vector | string scalar

Background color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the BackgroundColor property adds a corresponding `mlreportgen.dom.BackgroundColor` format object to the Style property. Setting the BackgroundColor property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

**Underline – Type of underline**

[] (default) | 'single' | 'none' | ...

Type of underline, specified as one of the values in this table.

Underline value	Description	Supported Output Types
'single'	Single line	All
'words'	Underline all characters except spaces	Word
'double'	Double line	Word
'thick'	Thick line	Word
'dotted'	Dotted line	Word
'dottedHeavy'	Thick dotted line	Word
'dash'	Dashed line	Word
'dashedHeavy'	Line with heavy dashes	Word
'dashLong'	Line with long dashes	Word
'dashLongHeavy'	Line with heavy long dashes	Word
'dotDash'	Dot-dash line	Word
'dashDotHeavy'	Heavy dash-dot line	Word
'dotDotDash'	Dot-dot-dash line	Word
'dashDotDotHeavy'	Line with heavy dashes with two dots between the dashes	Word
'wave'	Wavy line	Word
'wavyHeavy'	Heavy wavy	Word
'wavyDouble'	Double wavy line	Word
'none'	No underline	All

Setting the Underline property adds a corresponding `mlreportgen.dom.Underline` format object to the Style property. Setting the Underline property to an empty value removes the object.

To specify both the color and the type of the underline, do not set the Underline property. Instead, set the Style property to include an `mlreportgen.dom.Underline` format object that specifies the underline type and color.



**WhiteSpace — How to treat white space in text**

[] (default) | 'normal' | 'nowrap' | ...

How to treat white space in text, specified as one of the values in this table.

Value	Description	Supported Output Types
'normal'	For HTML and PDF, this value removes spaces at the beginning and the end of text. Multiple spaces in the text collapse to a single space.  For Word, this value removes spaces at the beginning and end of text.	All
'nowrap'	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
'pre'	Preserves white space. Text wraps only on line breaks. Acts like the <pre> tag in HTML.	HTML and PDF
'pre-line'	Sequences of white space collapse into a single white space. Text wraps when necessary and on line breaks.	HTML and PDF
'pre-wrap'	Preserves white space. Text wraps when necessary and on line breaks.	HTML and PDF
'preserve'	Same as 'pre'.	All

Setting the `WhiteSpace` property adds a corresponding `mlreportgen.dom.WhiteSpace` format object to the `Style` property. Setting the `WhiteSpace` property to an empty value removes the object.

**FontFamilyName — Name of font family for text**

[] | character vector | string scalar

Name of a font family to use for text, specified as a character vector or string scalar.

Setting the `FontFamilyName` property adds a corresponding `mlreportgen.dom.FontFamily` format object to the `Style` property. Setting the `FontFamilyName` property is the same as setting the `FontName` property of an `mlreportgen.dom.FontFamily` object. Setting the `FontFamilyName` property to an empty value removes the object.

To specify substitutions for the font, do not specify the font family by setting this property. Instead create and add an `mlreportgen.dom.FontFamily` object to the `Style` property.

Example: 'Courier New'

**FontSize — Font size for text**

[] (default) | character vector | string scalar

Font size to use for text, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '12pt' specifies twelve points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

Setting the `FontSize` property adds a corresponding `mlreportgen.dom.FontSize` format object to the `Style` property. Setting the `FontSize` property to an empty value removes the object.

Example: '12pt'

### Strike — Text strikethrough

[] (default) | 'none' | 'single' | 'double'

Text strikethrough, specified as one of these values:

- 'none' — No strikethrough
- 'single' — Single line
- 'double' — Double line (for Word reports only)

Setting the `Strike` property adds a corresponding `mlreportgen.dom.Strike` format object to the `Style` property for this document element. Setting the `Strike` property to an empty value removes the object.

### HAAlign — Horizontal alignment of paragraph

'center' | 'distribute' | ...

Horizontal alignment for this paragraph, relative to page margins or table cell borders, specified as one of the values in this table.

Value	Description	Supported Output Types
'center'	Center the paragraph	All
'distribute'	Distribute all characters equally	Word
'justify'	Align left side of paragraph on left side of page or table entry, and right side of paragraph on the right side of the page or table entry.	All
'KashidaHigh'	Use widest Kashida length.  Kashida is a type of justification used for some cursive scripts, such as Arabic and Persian.	Word
'KashidaLow'	Use lowest Kashida length	Word

Value	Description	Supported Output Types
'KashidaMedium'	Use medium Kashida length	Word
'left'	Align paragraph left	All
'right'	Align paragraph right	All
'ThaiDistribute'	Thai language justification	Word

Setting the `HAlign` property adds a corresponding `mlreportgen.dom.HAlign` format object to the `Style` property for this document element. Setting the `HAlign` property to an empty value removes the object.

### OuterLeftMargin — Left indentation for paragraph

[ ] | character vector | string scalar

Left indentation for this paragraph, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '0.5in' specifies one-half inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

The left indentation is the space between the left outer boundary of this paragraph and the left inner boundary of its container. This is equivalent to the left indentation property of a Microsoft Word paragraph.

Setting the `OuterLeftMargin` property adds a corresponding `mlreportGen.dom.OuterMargin` format object to the `Style` property for this document element. Setting the `OuterLeftMargin` property to an empty value removes the object.

To indent a paragraph from both the left and right margin of a page, do not set this property. Instead, create an `mlreportgen.dom.OuterMargin` that specifies the left and right indentations and add the object to the `Style` property of this paragraph.

### FirstLineIndent — Indentation amount for first line of paragraph

[ ] | character vector | string scalar

Indentation amount for the first line of this paragraph, specified as a character vector or string scalar that consists of number followed by an abbreviation for a unit of measurement. For example, '0.5in' specifies one-half inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas

- `pt` — points

The indentation amount is the amount by which to indent the first line of this paragraph relative to the succeeding lines. To create a hanging indent where all of the lines are indented except for the first line, use a negative number.

Setting the `FirstLineIndent` property adds a corresponding `mlreportgen.dom.FirstLineIndent` format object to the `Style` property for this document element. Setting the `FirstLineIndent` property to an empty value removes the object.

Example: `'0.5in'`

### **StyleName — Name of style for formatting paragraph**

`[]` | character vector | string scalar

Name of the style for formatting this paragraph, specified as a character vector or string scalar.

The style specified by the `StyleName` property must be defined in the stylesheet of the document or document part to which this paragraph is appended. The specified style defines the appearance of the paragraph in the output document, except for formats that are specified by the `Style` property of this paragraph. The format objects specified by the `Style` property override formats defined in the style.

### **Style — Formats that define paragraph style**

cell array of DOM format objects

Formats that define the style of this paragraph, specified as a cell array of DOM format objects. The formats override the corresponding formats defined by the stylesheet style specified by the `StyleName` property.

### **CustomAttributes — Custom attributes of paragraph**

`[]` | cell array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this paragraph, specified as a cell array of `mlreportgen.dom.CustomAttribute` objects. The output format must support the custom attributes.

### **Parent — Parent of paragraph**

document element object

Parent of paragraph, specified as a document element object. This property is read-only.

### **Children — Children of paragraph**

array of document element objects

Children of this paragraph, specified as an array of document element objects. This property is read-only.

### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the

object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

## **Methods**

### **Public Methods**

append Append content to paragraph  
clone Copy paragraph object

## **Examples**

### **Add Paragraphs**

Add a paragraph with text and another with an external link.

```
import mlreportgen.dom.*
doc = Document('mydoc','html');

p1 = Paragraph('This will be bold text');
p1.Bold = true;
link = ExternalLink('https://www.mathworks.com/', 'MathWorks');
p2 = Paragraph(link);
p2.BackgroundColor = 'yellow';
append(doc,p1);
append(doc,p2);

close(doc);
rptview('mydoc','html');
```

### **See Also**

mlreportgen.dom.Preformatted | mlreportgen.dom.HAlign |  
mlreportgen.dom.FontFamily | mlreportgen.dom.FontSize | mlreportgen.dom.Bold |  
mlreportgen.dom.Italic | mlreportgen.dom.Color | mlreportgen.dom.Strike |  
mlreportgen.dom.Underline | mlreportgen.dom.OuterMargin |  
mlreportgen.dom.BackgroundColor | mlreportgen.dom.CustomAttribute |  
mlreportgen.dom.FirstLineIndent | mlreportgen.dom.WhiteSpace

### **Topics**

“Add Content to a Report” on page 13-10  
“Report Formatting Approaches” on page 13-17

### **Introduced in R2014b**

## mlreportgen.dom.PDFPageFooter class

**Package:** mlreportgen.dom

Page footer definition for PDF document

### Description

Add a footer to the first page of a layout or to odd pages, even pages, or both.

### Construction

`pdfFooter = PDFPageFooter()` creates an empty page footer based on the default PDF template.

`pdfFooter = PDFPageFooter(pageType)` creates a page footer for the specified type of page, that is, odd, even, or first, based on the default PDF template.

`pdfFooter = PDFPageFooter(pageType, templatePath)` creates a page footer based on the specified template.

`pdfFooter = PDFPageFooter(pageType, templatePath, docPartTemplateName)` creates a page footer for the specified type of page, based on the specified document part template in the specified template.

`pdfFooter = PDFPageFooter(pageType, templateSrc, docPartTemplateName)` creates a page footer for the specified type of page, based on the specified document part template from the specified source. The source can be a document or a document part.

### Input Arguments

#### **pageType — Type of pages the footer appears on**

`[]` (default) | `default` | `first` | `even`

Type of pages the footer appears on, specified as one of these values:

- `default` — Footer for odd pages of the section, even pages if you do not specify an even-page footer, and first page if you do not specify a first-page footer.
- `first` — Footer for first page of a section.
- `even` — Footer for even pages of a section.

For example, to make different footers appear on odd pages and on even pages, define two footers. Set `pageType` to `default` for one and to `even` for the other.

#### **templatePath — Full path of footer template**

character vector

Full path of footer template, specified as a character vector.

#### **docPartTemplateName — Document part template name**

character vector

Name of this part's template if it is stored in a template specified by the `templatePath` or `templateSrc` argument, specified as a character vector.

**templateSrc — Document or document part that holds the document part template**

`mlreportgen.dom.Document` object | `mlreportgen.dom.DocumentPart` object

Document or document part object whose template contains the template for this document part, specified as an `mlreportgen.dom.Document` object for a document or an `mlreportgen.dom.DocumentPart` object for a document part.

**Output Arguments**

**pdfFooter — Page footer for PDF document**

`mlreportgen.dom.PDFPageFooter` object

Page footer for a PDF document, returned as an `mlreportgen.dom.PDFPageFooter` object.

**Properties**

**Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**CurrentPageLayout — Page footers ignore this property**

Not applicable

This property does not apply to page footers.

**CurrentHoleId — ID of current hole in document**

character vector

This read-only property is the hole ID of the current hole in this document.

**CurrentHoleType — Type of current hole**

'Inline' | 'Block'

Type of the current template hole, specified as 'Inline' or 'Block'.

- An inline hole is for document elements that a paragraph element can contain: Text, Image, LinkTarget, ExternalLink, InternalLink, CharEntity, AutoNumber.
- A block hole can contain a Paragraph, Table, OrderedList, UnorderedList, DocumentPart, or Group.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**PageType — Type of pages on which footer appears**

[] (default) | default | first | even

Type of page on which the footer appears, specified as one of these values:

- `default` — Footer for odd pages of the section, even pages if you do not specify an even-page footer, and first page if you do not specify a first-page footer.
- `first` — Footer for first page of a section.
- `even` — Footer for even pages in a section.

To have a footer appear on odd pages and on even pages, define two footers, one with `pageType` set to `default` and the other with `pageType` set to `even`.

### Parent — Parent of document element

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### TemplatePath — Path to template used for footer

character vector

Full path to the template to use for this footer, specified as a character vector.

## Methods

Use `DocumentPageFooter` methods as you use the corresponding `Document` methods.

Method	Purpose
<code>append</code>	Append one of these DOM objects to the footer: <ul style="list-style-type: none"> <li>• <code>CustomElement</code></li> <li>• <code>FormalTable</code></li> <li>• <code>Group</code></li> <li>• <code>ExternalLink</code></li> <li>• <code>Image</code></li> <li>• <code>InternalLink</code></li> <li>• <code>OrderedList</code></li> <li>• <code>Paragraph</code></li> <li>• <code>RawText</code></li> <li>• <code>Table</code></li> <li>• <code>Text</code></li> <li>• <code>UnorderedList</code></li> </ul>
<code>close</code>	Close the footer.



Method	Purpose
fill	Fill the template hole.
moveToNextHole	Move to the next template hole.
open	Open the footer.

## Examples

### Add a Footer to a PDF Document

This example defines first, even, and odd page footers in a PDF document. It inserts a page number in each footer, using a different alignment for each page type.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'pdf');
open(d);

% Create page footer objects for each type of page
% Assign a matrix of page footer objects to the current page layout
firstfooter = PDFPageFooter('first');
evenfooter = PDFPageFooter('even');
oddfooter = PDFPageFooter('default');
d.CurrentPageLayout.PageFooters = [firstfooter, evenfooter, oddfooter];

% Add title to first page footer
p = Paragraph('My Document Title');
p.HAlign = 'center';
append(d.CurrentPageLayout.PageFooters(1), p);

% Add page number to even page footer
% Align even page numbers left
pg2 = Page();
p2 = Paragraph();
p2.HAlign = 'left';
append(p2, pg2);
append(d.CurrentPageLayout.PageFooters(2), p2);

% Add page number to odd page footer
% Align odd page numbers right
pg3 = Page();
p3 = Paragraph();
p3.HAlign = 'right';
append(p3, pg3);
append(d.CurrentPageLayout.PageFooters(3), p3);

% Create several pages.
p = Paragraph('Hello World');
append(d, p);
p = Paragraph('Another page');
p.Style = {PageBreakBefore(true)};
append(d, p);
append(d, clone(p));
```

```
close(d);  
rptview(d.OutputPath);
```

### **See Also**

`mlreportgen.dom.PDFPageHeader` | `mlreportgen.dom.PDFPageLayout` |  
`mlreportgen.dom.DOCXPageHeader` | `mlreportgen.dom.DOCXPageFooter` |  
`mlreportgen.dom.DOCXPageLayout`

### **Topics**

“Create Page Footers and Headers” on page 13-148

### **Introduced in R2016a**

# mlreportgen.dom.PDFPageHeader class

**Package:** mlreportgen.dom

Page header definition for PDF document

## Description

Add a header to the first page of a section or to odd pages, even pages, or both.

## Construction

`pdfHeader = PDFPageHeader()` creates an empty page header based on the default PDF template.

`pdfHeaderer = PDFPageHeader(pageType)` creates a page header for the specified type of page, that is, odd, even, or first, based on the default PDF template.

`PageHeader = PDFPageHeader(pageType, templatePath)` creates a page header for the specified type of page based on the specified template.

`pdfHeader = PDFPageHeader(pageType, templatePath, docPartTemplateName)` creates a page header for the specified type of page, based on the specified document part template in the specified template.

`pdfHeader = PDFPageHeader(pageType, templateSrc, docPartTemplateName)` creates a page header for the specified type of page, based on the specified document part template used by the specified source. The source can be a document or a document part.

## Input Arguments

### **pageType** — Type of pages header appears on

`[]` (default) | `default` | `first` | `even`

Type of page header appears on, specified as one of these values:

- `default` — Header for odd pages of the section, even pages if you do not specify an even-page header, and first page if you do not specify a first-page header.
- `first` — Header for first page of a section.
- `even` — Header for even pages in a section.

For example, to have a blank header appear on the first page of a section and a different header appear on the other pages, define two headers, one with `pageType` set to `first` and the other with `pageType` set to `default`.

### **templatePath** — Full path of header template

character vector

Full path of footer template, specified as a character vector.

### **docPartTemplateName** — Document part template name

character vector

Name of this part's template if it is stored in a template specified by the `templatePath` or `templateSrc` argument, specified as a character vector.

**templateSrc — Document or document part that holds the document part template**

`mreportgen.dom.Document` object | `mreportgen.dom.DocumentPart` object

Document or document part object whose template contains the template for this document part, specified as an `mreportgen.dom.Document` object for a document or an `mreportgen.dom.DocumentPart` object for a document part.

**Output Arguments****pdfHeader — Page header for PDF document**

`mreportgen.dom.PDFPageHeader` object

Page header for a PDF document, returned as an `mreportgen.dom.PDFPageHeader` object.

**Properties****Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**CurrentHoleId — ID of current hole in document**

character vector

This read-only property is the hole ID of the current hole in this document.

**CurrentHoleType — Type of current hole**

'Inline' | 'Block'

Type of the current template hole, specified as 'Inline' or 'Block'.

- An inline hole is for document elements that a paragraph element can contain: Text, Image, LinkTarget, ExternalLink, InternalLink, CharEntity, AutoNumber.
- A block hole can contain a Paragraph, Table, OrderedList, UnorderedList, DocumentPart, or Group.

**CurrentPageLayout — Current page layout of this document**

`mreportgen.dom.DOCXPageLayout` object | `mreportgen.dom.PDFPageLayout` object

This property applies to Word and PDF documents. For Word documents, the value is a `DOCXPageLayout` object that specifies the current page layout. For PDF documents, the value is a `PDFPageLayout` object if the document currently specifies a page layout. For HTML documents, the value is always [].

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**PageType — Type of pages header appears on**

[] (default) | default | first | even

Type of page header appears on, specified as one of these values:

- `default` — Header for odd pages of the section, even pages if you do not specify an even-page header, and first page if you do not specify a first-page header.
- `first` — Header for first page of a section.
- `even` — Header for even pages in a section.

For example, to have a blank header appear on the first page and a different header appear on the other pages, define two headers, one with `pageType` set to `first` and the other with `pageType` set to `default`.

#### **Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

#### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

#### **TemplatePath — Full path to template for header**

character vector

Full path to template to use for this header, specified as a character vector.

## **Methods**

Use `PDFPageHeader` methods as you use the corresponding `Document` methods.

Method	Purpose
append	Append one of these DOM objects to the header: <ul style="list-style-type: none"> <li>• CustomElement</li> <li>• PDFPageLayout</li> <li>• FormalTable</li> <li>• Group</li> <li>• ExternalLink</li> <li>• Image</li> <li>• InternalLink</li> <li>• OrderedList</li> <li>• Paragraph</li> <li>• RawText</li> <li>• Table</li> <li>• Text</li> <li>• UnorderedList</li> </ul>
close	Close header.
fill	Fill template hole.
moveToNextHole	Move to next template hole.
open	Open header.

### See Also

[mlreportgen.dom.PDFPageFooter](#) | [mlreportgen.dom.PDFPageLayout](#) |  
[mlreportgen.dom.DOCXPageHeader](#) | [mlreportgen.dom.DOCXPageFooter](#) |  
[mlreportgen.dom.DOCXPageLayout](#)

### Topics

“Create Page Footers and Headers” on page 13-148

### Introduced in R2016a

# mlreportgen.dom.PDFPageLayout class

**Package:** mlreportgen.dom mlreportgen.dom

Page format and layout for section of PDF document

## Description

Use an mlreportgen.dom.PDFPageLayout object to define the page format, headers, and footers of a section of a PDF document.

The mlreportgen.dom.PDFPageLayout class is a handle class.

## Class Attributes

HandleCompatible	true
ConstructOnLoad	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

pageLayoutObj = mlreportgen.dom.PDFPageLayout() creates a PDFPageLayout object with default property on page 12-399 values.

## Properties

### PageHeaders — Page headers for this layout

array of mlreportgen.dom.PDFPageHeader objects

Page headers for this layout, specified as an array of mlreportgen.dom.PDFPageHeader objects. You can define up to three page headers for a layout, one each for:

- The first page of the section
- Even pages
- Odd pages

### PageFooters — Page footers for this layout

array of mlreportgen.dom.PDFPageFooter objects

Page footers for this layout, specified as an array of mlreportgen.dom.PDFPageFooter objects. You can define up to three page footers for a layout, one each for:

- The first page of the section
- Even pages
- Odd pages

**FirstPageNumber — Number of first page in section**

integer

Number of the first page in a section, specified as an integer.

---

**Note** Page numbers are rendered only on `mReportGen.dom.Page` objects that you append to the document or to any part of the document. To render page numbers in the page headers or footers, assign an `mReportGen.dom.PDFPageHeader` object to the `PageHeaders` property or an `mReportGen.dom.PDFPageFooter` object to the `PageFooters` property. Then append an `mReportGen.dom.Page` object to the `PDFPageHeader` or `PDFPageFooter` object. For example, see “Add and Customise Page Numbers in a PDF Document” on page 12-403. Alternatively, you can append `mReportGen.dom.Page` objects to the document itself, or to elements in the document, but the page number renders only on the part of the document where you append the `Page` object.

---

**PageNumberFormat — Type of page numbering to use**

character vector | string scalar

Type of page numbering to use, specified as one of the character vectors or string scalars in the table.

Value	Meaning	Applies To	
		DOCX	PDF
'a'	Lowercase alphabetic	✓	✓
'A'	Uppercase alphabetic	✓	✓
'i'	Lowercase Roman numerals	✓	✓
'I'	Uppercase Roman numerals	✓	✓
'n', 'N', '1', 'decimal'	Arabic numerals	✓	✓
'numberInDash'	Number with dashes on either side	✓	
'hebrew1'	Hebrew numerals	✓	
'hebrew2'	Hebrew alphabetic	✓	
'arabicAlpha'	Arabic alphabetic	✓	
'arabicAbjad'	Arabic abjad numerals	✓	
'thaiLetters'	Thai letters	✓	
'thaiNumbers'	Thai numerals	✓	
'thaiCounting'	Thai counting system	✓	



---

**Note** Page numbers are rendered only on `mlreportgen.dom.Page` objects that you append to the document or to any part of the document. To render page numbers in the page headers or footers, assign an `mlreportgen.dom.PDFPageHeader` object to the `PageHeaders` property or an `mlreportgen.dom.PDFPageFooter` object to the `PageFooters` property. Then append an `mlreportgen.dom.Page` object to the `PDFPageHeader` or `PDFPageFooter` object. For example, see “Add and Customise Page Numbers in a PDF Document” on page 12-403. Alternatively, you can append `mlreportgen.dom.Page` objects to the document itself, or to elements in the document, but the page number renders only on the part of the document where you append the `Page` object.

---

### Watermark — Watermark to apply to this section

`mlreportgen.dom.Watermark` object

Watermark to apply to this section, specified as an `mlreportgen.dom.Watermark` object.

### Hyphenation — Type of hyphenation

`false` | `true` | `' - '` | `' '`

Type of hyphenation, specified as one of these values:

- `true` — Enables hyphenation and uses `' - '` for the hyphenation character.
- `' - '` — Enables hyphenation and uses `' - '` for the hyphenation character.
- `' '` — Enables hyphenation and uses a space (`' '`) for the hyphenation character
- `false` — Disables hyphenation.

### PageMargins — Sizes of margins, header, footer, and gutter

`mlreportgen.dom.PageMargins` object

Sizes of the margins, header, footer, and gutter for this page layout, specified as an `mlreportgen.dom.PageMargins` object.

### PageSize — Size of pages in this layout

`mlreportgen.dom.PageSize` object

Size of pages in this layout, specified as an `mlreportgen.dom.PageSize` object.

### PageBorder — Page borders for this layout

`[]` (default) | `mlreportgen.dom.PageBorder` object

Page borders for this layout, specified as an `mlreportgen.dom.PageBorder` object.

### SectionBreak — Section break options

`'Next Page'` | `'Odd Page'` | `'Even Page'`

Section break options, specified as one of these character vectors or string scalars:

- `'Next Page'` — Start the section on the next page.
- `'Odd Page'` — Start the section on an odd page.
- `'Even Page'` — Start the section on an even page.

### StyleName — Ignored by page layouts

not applicable

This property does not apply to page layouts.

**Style — Formats to apply to layout**

array of format objects

Formats to apply to this layout, specified as an array of format objects. Formats that do not apply to a page layout are ignored.

**CustomAttributes — Custom attributes of document element**array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this document element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. The custom attributes must be supported by the output format.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Methods****Public Methods**

rotate	rotate( <code>pageLayoutObj</code> ) switches the orientation of pages between portrait and landscape. The method switches the values of the <code>Height</code> and <code>Width</code> properties of the <code>PageSize</code> object that is associated with the <code>PageSize</code> property of <code>pageLayoutObj</code> . The method also switches the value of the <code>Orientation</code> property of the <code>PageSize</code> object between 'portrait' and 'landscape'.
--------	---

## Examples

### Change Page Margins of a Document Section

Use the `CurrentPageLayout` property of a document to access the `PDFPageLayout` object that represents the layout of the document. Change the left and right margins of the layout by setting the `Left` and `Right` properties of the `mlreportgen.dom.PageMargins` object used by the `PDFPageLayout` object.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'pdf');

open(d);
pageLayoutObj = d.CurrentPageLayout;
pageLayoutObj.PageMargins.Left = '2in';
pageLayoutObj.PageMargins.Right = '2in';
p = Paragraph('Hello World');
append(d,p);

close(d);
rptview(d);
```

### Change the Page Orientation

By default, a `PDFPageLayout` object specifies a page with an 11-inch height, 8.5-half inch width, and portrait orientation. To change the orientation to landscape, use the `rotate` method of the `PDFPageLayout` object. The method updates the property values of the associated `mlreportgen.dom.PageSize` object so that the height is 8.5 inches, the width is 11 inches, and the orientation is landscape.

```
import mlreportgen.dom.*;
d = Document('myreport', 'pdf');
open(d);

pageLayoutObj = d.CurrentPageLayout;
rotate(pageLayoutObj);

append(d, 'This document has landscape pages');
close(d);
rptview(d);
```

### Add and Customise Page Numbers in a PDF Document

This example shows how to add and customize page numbers in the footer section of a generated PDF document.

Import this package so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
```

Create an `mlreportgen.dom.Document` object of type PDF with a title.

```
document = Document("my_pdf_w_page_numbers_document", "pdf");
heading = Heading(1, "Customize Page Numbers Example");
append(document, heading);
```

Add content to the document.

```
append(document, LineBreak);
for pageInd = 1:5
    append(document, sprintf("Content of page number %i", pageInd));
    append(document, PageBreak);
end
append(document, Paragraph("Content of last page"));
```

Add a page footer to the document by assigning an `mlreportgen.dom.PDFPageFooter` object to the `PageFooters` property of the current page layout. Then set the page number format of the current page layout to uppercase Roman numerals.

```
curLayout = document.CurrentPageLayout;
curLayout.PageFooters = PDFPageFooter();
curLayout.PageNumberFormat = "I";
```

Append an `mlreportgen.dom.Page` object to the page footer object.

```
append(curLayout.PageFooters, Page());
```

Close and view the document.

```
close(document);
rptview(document);
```

## See Also

`mlreportgen.dom.DocumentPart` | `mlreportgen.dom.PDFPageFooter` |  
`mlreportgen.dom.PDFPageHeader` | `mlreportgen.dom.PageSize` |  
`mlreportgen.dom.PageMargins` | `mlreportgen.dom.PageBorder` |  
`mlreportgen.dom.PageNumber` | `mlreportgen.dom.PageRawFormat` |  
`mlreportgen.dom.Watermark` | `mlreportgen.dom.Page`

## Topics

"Report Formatting Approaches" on page 13-17

## Introduced in R2016a

# mlreportgen.dom.Preformatted class

**Package:** mlreportgen.dom

**Superclasses:** mlreportgen.dom.Paragraph

Preformatted block of text

## Description

Use objects of the `mlreportgen.dom.Preformatted` class to preserve the white-space formatting of text. White space-formatting uses spaces, line feeds, and monospace fonts to render text on multiple lines and to control indentation. For example, use an `mlreportgen.dom.Preformatted` object to add program code to a report.

If you do not set the `FontFamilyName` or `Whitespace` properties, the DOM API uses the default values for the report type (HTML, Microsoft Word, or PDF) so that white-space formatting is preserved. You can override the default values by setting the `FontFamilyName` or `Whitespace` properties.

The `mlreportgen.dom.Preformatted` class is a `handle` class.

## Class Attributes

<code>ConstructOnLoad</code>	<code>true</code>
<code>HandleCompatible</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`preObj = mlreportgen.dom.Preformatted()` creates an empty preformatted paragraph.

`preObj = mlreportgen.dom.Preformatted(text)` creates a preformatted paragraph that contains the specified text.

`preObj = mlreportgen.dom.Preformatted(text, styleName)` creates a preformatted paragraph that has the specified style. The style specified by the `styleName` property must be defined in the template used for the document element to which this paragraph is appended.

`preObj = mlreportgen.dom.Preformatted(docElementObj)` creates a paragraph that contains the document element specified by `docElementObj`.

### Input Arguments

#### text — Paragraph text

character vector | string scalar

Paragraph text, specified as a character vector or string scalar.

**styleName — Style for paragraph**

character vector | string scalar

Style for this paragraph, specified as a character vector or string scalar. The style must be defined in the template used to create the document to which this paragraph is appended.

**docElementObj — Document element to include in paragraph**

DOM object

Document element (DOM object) to include in this paragraph. You can specify these DOM objects:

- `mlreportgen.dom.ExternalLink`
- `mlreportgen.dom.Image`
- `mlreportgen.dom.InternalLink`
- `mlreportgen.dom.LinkTarget`
- `mlreportgen.dom.Text`

**Properties****OutlineLevel — Outline level of paragraph**

[] (default) | integer

Outline level of the paragraph, specified as an integer. Setting the `OutlineLevel` property causes this paragraph to be included in automatically generated outlines, such as a table of contents. The value specifies the level of the paragraph in the outline. For example, to make a paragraph appear at the top level in an outline, set the `OutlineLevel` property to 1.

Setting the `OutlineLevel` property adds a corresponding `mlreportgen.dom.OutlineLevel` format object to the `Style` property. Setting the `OutlineLevel` property to an empty value removes the object.

**Bold — Whether to use bold for text**

[] (default) | true or 1 | false or 0

Whether to use bold for text, specified as a numeric or logical 1 (true) or 0 (false). To make text bold, set `Bold` to true or 1.

Setting the `Bold` property adds a corresponding `mlreportgen.dom.Bold` format object to the `Style` property. Setting the `Bold` property to an empty value removes the object.

**Italic — Whether to use italic for text**

[] (default) | true or 1 | false or 0

Whether to use italic for text, specified as a numeric or logical 1 (true) or 0 (false). To make text italic, set `Italic` to true or 1.

Setting the `Italic` property adds a corresponding `mlreportgen.dom.Italic` format object to the `Style` property. Setting the `Italic` property to an empty value removes the object.

**Color — Text color**

[] (default) | character vector | string scalar

Text color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `Color` property adds a corresponding `mlreportgen.dom.Color` format object to the `Style` property. Setting the `Color` property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

### BackgroundColor — Background color

[] (default) | character vector | string scalar

Background color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form #RRGGBB.

Setting the `BackgroundColor` property adds a corresponding `mlreportgen.dom.BackgroundColor` format object to the `Style` property. Setting the `BackgroundColor` property to an empty value removes the object.

Example: 'blue'

Example: '#0000ff'

### Underline — Type of underline

[] (default) | 'single' | 'none' | ...

Type of underline, specified as one of the values in this table.

Underline value	Description	Supported Output Types
'single'	Single line	All
'words'	Underline all characters except spaces	Word
'double'	Double line	Word
'thick'	Thick line	Word
'dotted'	Dotted line	Word
'dottedHeavy'	Thick dotted line	Word
'dash'	Dashed line	Word
'dashedHeavy'	Line with heavy dashes	Word
'dashLong'	Line with long dashes	Word
'dashLongHeavy'	Line with heavy long dashes	Word
'dotDash'	Dot-dash line	Word
'dashDotHeavy'	Heavy dash-dot line	Word
'dotDotDash'	Dot-dot-dash line	Word
'dashDotDotHeavy'	Line with heavy dashes with two dots between the dashes	Word
'wave'	Wavy line	Word

Underline value	Description	Supported Output Types
'wavyHeavy'	Heavy wavy	Word
'wavyDouble'	Double wavy line	Word
'none'	No underline	All

Setting the Underline property adds a corresponding `mlreportgen.dom.Underline` format object to the Style property. Setting the Underline property to an empty value removes the object.

To specify both the color and the type of the underline, do not set the Underline property. Instead, set the Style property to include an `mlreportgen.dom.Underline` format object that specifies the underline type and color.

### WhiteSpace — How to treat white space in text

[] (default) | 'normal' | 'nowrap' | ...

How to treat white space in text, specified as one of the values in this table.

Value	Description	Supported Output Types
'normal'	For HTML and PDF, this value removes spaces at the beginning and the end of text. Multiple spaces in the text collapse to a single space.  For Word, this value removes spaces at the beginning and end of text.	All
'nowrap'	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
'pre'	Preserves white space. Text wraps only on line breaks. Acts like the <code>&lt;pre&gt;</code> tag in HTML.	HTML and PDF
'pre-line'	Sequences of white space collapse into a single white space. Text wraps when necessary and on line breaks.	HTML and PDF
'pre-wrap'	Preserves white space. Text wraps when necessary and on line breaks.	HTML and PDF
'preserve'	Same as 'pre'.	All

Setting the WhiteSpace property adds a corresponding `mlreportgen.dom.WhiteSpace` format object to the Style property. Setting the WhiteSpace property to an empty value removes the object.

### FontFamilyName — Name of font family for text

[] | character vector | string scalar



Name of a font family to use for text, specified as a character vector or string scalar.

Setting the `FontFamilyName` property adds a corresponding `mlreportgen.dom.FontFamily` format object to the `Style` property. Setting the `FontFamilyName` property is the same as setting the `FontName` property of an `mlreportgen.dom.FontFamily` object. Setting the `FontFamilyName` property to an empty value removes the object.

To specify substitutions for the font, do not specify the font family by setting this property. Instead create and add an `mlreportgen.dom.FontFamily` object to the `Style` property.

Example: 'Courier New'

### FontSize — Font size for text

[] (default) | character vector | string scalar

Font size to use for text, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '12pt' specifies twelve points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

Setting the `FontSize` property adds a corresponding `mlreportgen.dom.FontSize` format object to the `Style` property. Setting the `FontSize` property to an empty value removes the object.

Example: '12pt'

### Strike — Text strikethrough

[] (default) | 'none' | 'single' | 'double'

Text strikethrough, specified as one of these values:

- 'none' — No strikethrough
- 'single' — Single line
- 'double' — Double line (for Word reports only)

Setting the `Strike` property adds a corresponding `mlreportgen.dom.Strike` format object to the `Style` property for this document element. Setting the `Strike` property to an empty value removes the object.

### HALign — Horizontal alignment of paragraph

'center' | 'distribute' | ...

Horizontal alignment for this paragraph, relative to page margins or table cell borders, specified as one of the values in this table.

Value	Description	Supported Output Types
'center'	Center the paragraph	All

Value	Description	Supported Output Types
'distribute'	Distribute all characters equally	Word
'justify'	Align left side of paragraph on left side of page or table entry, and right side of paragraph on the right side of the page or table entry.	All
'KashidaHigh'	Use widest Kashida length.  Kashida is a type of justification used for some cursive scripts, such as Arabic and Persian.	Word
'KashidaLow'	Use lowest Kashida length	Word
'KashidaMedium'	Use medium Kashida length	Word
'left'	Align paragraph left	All
'right'	Align paragraph right	All
'ThaiDistribute'	Thai language justification	Word

Setting the `HALign` property adds a corresponding `mreportgen.dom.HALign` format object to the `Style` property for this document element. Setting the `HALign` property to an empty value removes the object.

### FirstLineIndent — Indentation amount for first line of paragraph

[ ] | character vector | string scalar

Indentation amount for the first line of this paragraph, specified as a character vector or string scalar that consists of number followed by an abbreviation for a unit of measurement. For example, '0.5in' specifies one-half inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

The indentation amount is the amount by which to indent the first line of this paragraph relative to the succeeding lines. To create a hanging indent where all of the lines are indented except for the first line, use a negative number.

Setting the `FirstLineIndent` property adds a corresponding `mreportgen.dom.FirstLineIndent` format object to the `Style` property for this document element. Setting the `FirstLineIndent` property to an empty value removes the object.

Example: '0.5in'

### StyleName — Name of style for formatting paragraph

[ ] | character vector | string scalar

Name of the style for formatting this paragraph, specified as a character vector or string scalar.

The style specified by the `StyleName` property must be defined in the stylesheet of the document or document part to which this paragraph is appended. The specified style defines the appearance of the paragraph in the output document, except for formats that are specified by the `Style` property of this paragraph. The format objects specified by the `Style` property override formats defined in the style.

### **Style — Formats that define paragraph style**

cell array of DOM format objects

Formats that define the style of this paragraph, specified as a cell array of DOM format objects. The formats override the corresponding formats defined by the stylesheet style specified by the `StyleName` property.

### **CustomAttributes — Custom attributes of document element**

array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this document element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. The custom attributes must be supported by the output format.

### **Parent — Parent of paragraph**

document element object

Parent of paragraph, specified as a document element object. This property is read-only.

### **Children — Children of paragraph**

array of document element objects

Children of this paragraph, specified as an array of document element objects. This property is read-only.

### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

## **Methods**

### **Public Methods**

`append` Append content to paragraph  
`clone` Copy paragraph object

## Examples

### Format Code in a Report

Format code in a report by using an `mlreportgen.dom.Preformatted` object. The white space and newline characters are preserved in the output. The font of the code is monospace.

```
import mlreportgen.dom.*
import mlreportgen.report.*
rpt = Report('myReport', 'docx');

p = Paragraph("This is my code:");
pr = Preformatted("  a = 1;" + newline + "  b = 2;" + newline + "  c = a + b;");

add(rpt, p);
add(rpt, pr);

close(rpt);
rptview(rpt);
```

Here is the code in the generated report:

This is my code:

```
a = 1;
b = 2;
c = a + b;
```

### See Also

`mlreportgen.dom.Paragraph` | `mlreportgen.dom.HAlign` | `mlreportgen.dom.FontFamily` |  
`mlreportgen.dom.FontSize` | `mlreportgen.dom.Bold` | `mlreportgen.dom.Italic` |  
`mlreportgen.dom.Color` | `mlreportgen.dom.Strike` | `mlreportgen.dom.Underline` |  
`mlreportgen.dom.OuterMargin` | `mlreportgen.dom.BackgroundColor` |  
`mlreportgen.dom.CustomAttribute` | `mlreportgen.dom.FirstLineIndent` |  
`mlreportgen.dom.WhiteSpace`

### Topics

“Add Content to a Report” on page 13-10

“Report Formatting Approaches” on page 13-17

### Introduced in R2020a

# mlreportgen.dom.ProgressMessage class

**Package:** mlreportgen.dom

Progress message

## Description

Create a progress message with the specified text originating from the specified source object.

## Construction

`progressMsgObj = ProgressMessage(text, sourceDOMObject)` creates a progress message with the specified text, originating from the specified source object.

### Input Arguments

**text — Message text**

character vector

The text to display for the message.

**source — DOM object to from which message originates**

a DOM object

The DOM object from which the message originates.

### Output Arguments

**progressMsgObj — Progress message**

mlreportgen.dom.ProgressMessage object

Progress message, represented by an mlreportgen.dom.ProgressMessage object.

## Properties

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Source — Source DOM object message originates from**

a DOM object

Source DOM object from which the message originates.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Text — Text of message

character vector

Message text, specified as a character vector.

## Methods

Method	Purpose
formatAsHTML	Wrap message in HTML tags.
formatAsText	Format message as text.
passesFilter	Determine if message passes filter.

## Examples

### Create a Progress Message

Create the report document.

```
import mlreportgen.dom.*;
d = Document('test','html');
```

Create a message dispatcher.

```
dispatcher = MessageDispatcher.getTheDispatcher;
l = addlistener(dispatcher,'Message',...
    @(src, evtdata) disp(evtdata.Message.formatAsText));
```

Dispatch the message.

```
open(d);
```

```
dom.Document:2502 opening
dom.Document:2502 parsing template "B:/matlab/toolbox/shared/mlreportgen/dom/resources/templates,
dom.Document:2502 appended dom.TemplateText:2518
dom.Document:2502 appended dom.TemplateText:2521
dom.Document:2502 appended dom.TemplateText:2524
dom.Document:2502 moved to hole "#start#"
```

```
dispatch(dispatcher,ProgressMessage('starting chapter',d));
```

```
dom.Document:2502 starting chapter
```

Add report content.

```
p = Paragraph('Chapter ');
p.Tag = 'chapter title';
p.Style = { CounterInc('chapter'),...
    CounterReset('table'),WhiteSpace('pre') };
```

```
append(p, AutoNumber('chapter'));  
append(d,p);
```

```
dom.Document:2502 appended chapter title
```

Run report and delete the listener.

```
close(d);
```

```
dom.Document:2502 appended dom.TemplateText:2539  
dom.Document:2502 moved to hole "#end#"  
dom.Document:2502 closed
```

```
rptview('test','html');
```

Delete the listener to avoid duplicate reporting of message objects during a MATLAB® session.

```
delete(l);
```

## See Also

dispatch

## Topics

“Display Progress and Debugger Messages” on page 13-115

## mlreportgen.dom.RawText class

**Package:** mlreportgen.dom

Word XML or HTML markup to insert in document

### Description

Word XML or HTML markup to insert in a document.

### Construction

`text = RawText()` creates an empty `RawText` object.

You can append a `RawText` object only to a `Document` object. For a Word document, the markup specified by the `DOCXText` property is included in the document. For an HTML document, the value of the `HTMLText` property is included. In either case, the markup must be valid Word XML or HTML markup, respectively, that can be validly inserted in the body element of the output document. If you insert invalid markup in a Microsoft Word document, Word may be unable to open the document.

`text = RawText(htmlMarkup)` creates a `RawText` object containing the specified HTML markup.

`text = RawText(markup,doctype)` creates a `RawText` object containing markup of the specified document type (HTML or Word).

### Input Arguments

#### **htmlMarkup — HTML markup code**

character vector

HTML markup, specified as a character vector. To improve the readability of your report document, consider assigning the markup to a variable. Then use the variable as an input argument, as shown in the example below.

#### **markup — Word XML or HTML markup code**

character vector

Word XML markup or HTML markup, specified as a character vector. For a Word document, the markup must be valid Word XML markup that can be inserted into the `w:body` element. To improve the readability of your report document, consider assigning the markup to a variable. Then use the variable as an input argument, as shown in the example below.

#### **doctype — Type of markup to use**

'html' | 'docx'

Type of markup to use, specified as a character vector.

### Output Arguments

#### **rawText — Word XML or HTML markup to insert in document**

mlreportgen.dom.RawText object



Word XML or HTML markup to insert in document, represented by an `mlreportgen.dom.RawText` object.

## Properties

### **DOCXText — Text to output to Word document**

character vector

Word XML markup, specified as a character vector. The value of this property is included in a Word document. The markup must be valid Word XML markup that can be inserted into the `w:body` element of a Word document.

### **HTMLText — Text to output to HTML document**

character vector

HTML markup, specified as a character vector. The value of this property is included in an HTML document. The text must be valid HTML markup that can be inserted into the `body` element of an HTML document.

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Examples

### **Add HTML Markup**

Assign HTML markup to a variable and use that variable to create a `RawText` object to append to a document.

```
import mlreportgen.dom.*;
d = Document('test','html');

script = [ ...
    '<script>' ...
    'document.write("Hello World!")' ...
```

```
        '</script>' ...  
    ];  
    append(d,RawText(script));  
  
    close(d);  
    rptview('test','html');
```

### See Also

mlreportgen.dom.CustomAttribute

### Topics

“Add Content to a Report” on page 13-10

# mlreportgen.dom.RepeatAsHeaderRow class

**Package:** mlreportgen.dom

Repeat table row

## Description

Use objects of the `mlreportgen.dom.RepeatAsHeaderRow` class to specify whether the table row repeats on each page when the table flows across multiple pages. You can use this object only with Microsoft Word documents.

## Construction

`repeatAsHeaderRowObj = RepeatAsHeaderRow()` repeats the table row on each page when the table flows across multiple pages.

`repeatAsHeaderRowObj = RepeatAsHeaderRow(onOff)` repeats the table row on each page when `onOff` is `true`.

---

**Note** You can use this object for the first row of the table and any immediately consecutive rows. If a row is not set to repeat over pages, then successive rows cannot be set to repeat over pages. If you apply this object to rows that follow non-repeating rows, this object is ignored.

---

## Input Arguments

### **onOff** — Whether table row repeats on each page

`true` or `1` | `false` or `0`

Whether the table row repeats on each page, specified as a numeric or logical `1` (`true`) or `0` (`false`), where:

- `true` or `1` — The table row repeats on each page when the table flows across multiple pages.
- `false` or `0` — The table row does not repeat.

Data Types: `logical`

## Properties

### **Id** — ID for this document element

`character vector` | `string scalar`

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **Tag** — Tag for this document element

`character vector` | `string scalar`

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Value — Whether table row repeats on each new page

true or 1 | false or 0

Whether the table row repeats on each page, specified as a numeric or logical 1 (true) or 0 (false), where:

- true or 1 — The table row repeats on each new page when a table flows across multiple pages.
- false or 0 — The table row does not repeat.

Data Types: logical

## Examples

### Repeat Table Rows

Create an `mlreportgen.dom.Document` of type docx.

```
import mlreportgen.dom.*;
document = Document("Repeat_Rows_Doc", "docx");
```

Create an `mlreportgen.dom.Table` with one column and 30 rows of `mlreportgen.dom.Text` objects, then set the text of the table rows.

```
append(document, "Table 1:");
table1 = Table(arrayfun(@(x) Text(), ones(30,1)));
table1.entry(1,1).Children(1).Content = "Header 1, does not repeat across pages";
table1.entry(2,1).Children(1).Content = "Header 2, does not repeat across pages";
for rowInd = 3:30
    table1.entry(rowInd,1).Children(1).Content = sprintf("Row %i of Table 1", rowInd);
end
```

Set the style for the table borders and separators, then append the table to the document.

```
table1.Style = [table1.Style, {Border("solid"), RowSep("solid")});
append(document, table1);
```

Create a second table with one column and 30 rows of `Text` objects, then set the text of the table rows.

```
append(document, LineBreak());
append(document, LineBreak());
append(document, "Table 2:");
table2 = Table(arrayfun(@(x) Text(), ones(30,1)));
table2.entry(1,1).Children(1).Content = "Header 1, repeats across pages";
table2.entry(2,1).Children(1).Content = "Header 2, repeats across pages";
for rowInd = 3:30
    table2.entry(rowInd,1).Children(1).Content = sprintf("Row %i of Table 2", rowInd);
end
```

Set the first two rows of the table to repeat across pages. Then set the style for the table borders and separators, and append the table to the document.

```
table2.row(1).Style = [table2.row(1).Style, {RepeatAsHeaderRow(true)}];
table2.row(2).Style = [table2.row(2).Style, {RepeatAsHeaderRow(true)}];
table2.Style = [table2.Style, {Border("solid"), RowSep("solid")});
append(document, table2);
```

Close the document to generate the report, then open the report.

```
close(document);  
rptview(document);
```

## **See Also**

[mlreportgen.dom.AllowBreakAcrossPages](#) | [mlreportgen.dom.TableHeader](#)

## **Topics**

“Create Informal Tables” on page 13-64

“Report Formatting Approaches” on page 13-17

## mlreportgen.dom.ResizeToFitContents class

**Package:** mlreportgen.dom

Allow table to resize its columns

### Description

Specifies whether a table resizes its columns to fit content.

The mlreportgen.dom.ResizeToFitContents class is a handle class.

### Creation

#### Description

resizeToFitContentsObj = ResizeToFitContents() allows a table to resize its columns to fit their contents.

resizeToFitContentsObj = ResizeToFitContents(tf) allows a table to resize its columns to fit their contents, if tf is true.

#### Input Arguments

##### value — Allow table to resize its columns

logical value

A setting of true (or 1) allows a table to resize its columns to fit their contents. A setting of false (or 0) causes the content to wrap.

Data Types: logical

### Properties

#### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

#### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Value — Allow table to resize its columns**

logical value

A setting of `true` (or `1`) allows a table to resize its columns to fit their contents. A setting of `false` (or `0`) causes the content to wrap.

If you set this property to `true` and also have set a column width using `mlreportgen.dom.Width`, the resulting column width depends on the output format. For PDF output, the table uses the specified column width and ignores the `ResizeToFitContents` setting. For all other output formats, `ResizeToFitContents` overrides the column width settings.

Data Types: `logical`**Examples****Resize Table Columns to Fit Content**

Create a table and specify to resize the column widths to fit the widest table entry in the column.

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);

append(d,Heading(1,'Table 1'));
table1 = Table(arrayfun(@(x) Text(), ones(4))); % Create a table with text entries
table1.entry(1,2).Children(1).Content = 'MathWorks';

table1.Style = {ResizeToFitContents(true),Width('lin'), ...
    Border('solid'),RowSep('solid'),ColSep('solid')};

append(d,table1);
```

Create a second table, but do not have the columns resize to fit content.

```
append(d,Heading(1,'Table 2'));
table2 = Table(arrayfun(@(x) Text(), ones(4))); % Create a table with text entries
table2.entry(1,2).Children(1).Content = 'MathWorks';

table2.Style = {ResizeToFitContents(false),Width('lin'), ...
    Border('solid'), RowSep('solid'),ColSep('solid')};

append(d,table2);
```

Run the report.

```
close(d);
rptview(d.OutputPath,doctype);
```

**See Also**

`mlreportgen.dom.TableColSpec` | `mlreportgen.dom.TableColSpecGroup` |  
`mlreportgen.dom.Table` | `mlreportgen.dom.FormalTable`

**Topics**

“Report Formatting Approaches” on page 13-17

**Introduced in R2014b**

## mlreportgen.dom.RowHeight class

**Package:** mlreportgen.dom

Height of table row

### Description

Specifies the height of a table row.

This format object enables you to specify an exact (fixed) row height in Microsoft Word output. If the row content does not fit in the specified height, Word truncates the content to preserve the specified height. For PDF and HTML output, the behavior of the `mlreportgen.dom.RowHeight` format object is the same as the behavior of the `mlreportgen.dom.Height` format object. With both format objects, PDF and HTML documents treat the specified height as a minimum to be adjusted upward to accommodate content. If you do not need to specify an exact height, you can use either `RowHeight` or `Height` to specify the height.

### Construction

`rowHeightObj = RowHeight()` specifies a row that is exactly one inch high.

`rowHeightObj = RowHeight(height)` sets a row to exactly the specified height.

`rowHeightObj = RowHeight(height, heightType)` sets a row to be exactly the specified height or at least the specified height, depending on the value of `heightType`.

### Output Arguments

**rowHeightObj — Height of table row**

`mlreportgen.dom.RowHeight` object

Row height, represented by an `mlreportgen.dom.RowHeight` object.

### Properties

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.



**Type — Type of row height**

'exact' | 'atleast'

Type of row height measurement, specified as one of these values:

- 'exact' — Microsoft Word generates a row of the specified height and truncates content that does not fit. HTML and PDF viewers create a row of at least the specified height and adjust the row height to accommodate the content.
- 'atleast' — Word, HTML and PDF viewers create a row of at least the specified height and adjust the row height to accommodate the content

**Value — Height of table row**

'lin' (default)

Height of table row in the form valueUnits, where Units is an abbreviation for the units. These abbreviations are valid:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Examples****Specify Table Row Heights**

Create a table with two rows. The first row has at least the specified height and the second has a fixed maximum height.

```
import mlreportgen.dom.*;
d = Document('myTableDoc', 'docx');

t = Table(2);
t.Style = {Border('solid'), RowSep('solid'), ColSep('solid')};
t.Width = 'lin';
r1 = TableRow();
r1.Style = {RowHeight('.25in', 'atleast')};
append(r1, TableEntry(...
    'This row can expand beyond .25 inches'));
append(r1, TableEntry('x'));

r2 = TableRow();
r2.Style = {RowHeight('.25in', 'exact')};
append(r2, TableEntry...
    ('Truncated text because height is fixed'));
append(r2, TableEntry('x'));

append(t, r1);
append(t, r2);
append(d, t);
```

```
close(d);  
rptview('myTableDoc', 'docx');
```

### **See Also**

`mlreportgen.dom.TableRow`

### **Topics**

“Create Informal Tables” on page 13-64

## mlreportgen.dom.RowSep class

**Package:** mlreportgen.dom

Draw lines between table rows

### Description

Draw lines (separators) between table rows.

### Construction

`rowSepObj = RowSep()` creates unspecified row separators.

`rowSepObj = RowSep(style)` creates a row separator of the specified style.

`rowSepObj = RowSep(style,color)` creates a row separator having the specified style and color.

`rowSepObj = RowSep(style,color,width)` creates a row separator having the specified style, color, and width.

### Input Arguments

#### **style** — Line style of table row separator

character vector

Line style of the table row separator, specified as one of these values.

Value	Applies To	
	Word	HTML and PDF
'dashed'	✓	✓
'dashdotstroked'	✓	
'dashsmallgap'	✓	
'dotted'	✓	✓
'dotdash'	✓	
'dotdotdash'	✓	
'double'	✓	✓
'doublewave'	✓	
'inset'	✓	✓
'none'	✓	✓

Value	Applies To	
	Word	HTML and PDF
'outset'	✓	✓
'single'	✓	
'solid'		✓
'thick'	✓	
'thickthinlargegap'	✓	
'thickthinmediumgap'	✓	
'thickthinsmallgap'	✓	
'thinthicklargegap'	✓	
'thinthickmediumgap'	✓	
'thinthicksmallgap'	✓	
'thinthickthinlargegap'	✓	
'thinthickthinmediumgap'	✓	
'thinthickthinsmallgap'	✓	
'threedemboss'	✓	
'threedengrave'	✓	
'triple'	✓	
'wave'	✓	

**color** — Color of table row separator

character vector

Color of the table row separator, specified as a color, such as 'red' or a hexadecimal RGB value, such as '#0000ff'.

**width** — Width of table row separator

character vector

Width of table row separator in the form valueUnits where Units is an abbreviation for the units. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters

- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

## Output Arguments

### **rowSepObj** — Draw lines between table rows

`mlreportgen.dom.RowSep` object

Table rows separator lines, represented by an `mlreportgen.dom.RowSep` object.

## Properties

### **Color** — Text color

`[]` (default) | character vector | string scalar

Text color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form `#RRGGBB`.

Setting the `Color` property adds a corresponding `mlreportgen.dom.Color` format object to the `Style` property. Setting the `Color` property to an empty value removes the object.

Example: `'blue'`

Example: `'#0000ff'`

### **Id** — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **Style** — Line style of table row separator

character vector

Line style for the row separator. See the description of the `style` input argument for a list of possible values.

### **Tag** — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### **Width** — Table row separator width

character vector

Width of the table row separator in the form `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

## Examples

### Use Table Row Separators

Define the row separator as part of the `Style` property definition for the table.

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);

t = Table(magic(5));
t.Style = {Border('inset','crimson','6pt'), ...
    ColSep('double','DarkGreen','3pt'), ...
    RowSep('double','Gold','3pt'), ...
    Width('50%')};

t.TableEntriesInnerMargin = '6pt';
t.TableEntriesHAlign = 'center';
t.TableEntriesVAlign = 'middle';
append(d,t);

close(d);
rptview(d.OutputPath);
```

### See Also

`mlreportgen.dom.Border` | `mlreportgen.dom.ColSep` | `mlreportgen.dom.TableRow` |  
`mlreportgen.dom.TableEntry`

### Topics

“Create Informal Tables” on page 13-64

# mlreportgen.dom.ScaleToFit class

**Package:** mlreportgen.dom

Scale image to fit page or table entry

## Description

Specifies whether to scale an image to fit between the margins of a Microsoft Word or PDF page or table entry.

To use this format to scale an image to fit a table entry, you must specify the entry height and width by including either:

- An mlreportgen.dom.Height or mlreportgen.dom.Width object in the entry Style property
- A Height or Width object in the parent table TableEntriesStyle property
- A Height object in the Style property of the parent table or table section
- A Width object in the Style property of the parent row

The mlreportgen.dom.ScaleToFit class is a handle class.

## Creation

### Description

scaleToFitObj = ScaleToFit() scales an image to fit between the margins of a page.

scaleToFitObj = ScaleToFit(value) scales an image if value is true.

### Input Arguments

#### value — Scale image to fit page

true | false

A setting of true (or 1) scales the image to fit between the margins of the page or within a table entry. A setting of false (or 0) does not scale the image.

## Properties

#### Value — Scale image to fit page

true (default) | false

The possible values are:

- true or 1— Scale image to fit between the margins or in table entry.
- false or 0 — Do not scale image.

#### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Examples

### Scale Image to Fit Margins and Table Entry

This example inserts an image in a paragraph and in a table entry using the ScaleToFit property on the image. The table entry uses a Height and Width property. You need at least one of these properties on the entry or inherited from the row, section, or parent table.

```
import mlreportgen.dom.*

d = Document('Scale to Fit Example','pdf');
open(d);

% Insert explanatory text in report
p = Paragraph(['Set the image style to ScaletToFit with ',...
    'img.Style = {ScaleToFit(true)}']);
append(d,p);

% Create the image object and set ScaleToFit property
img = Image(which('ngc6543a.jpg'));
img.Style = {ScaleToFit};
append(d,img);

% Explanatory text
p = Paragraph(['Scale image to fit the table cell, Set the ',...
    'height and width on the table with:']);
p.Style = {PageBreakBefore};
append(d,p);

% Create the table, setting height and width
% Create the image object and set ScaleToFit property
append(d,'table.entry(1,1).Style = {Height('lin'), Width('lin')}');
img = Image(which('ngc6543a.jpg'));
img.Style = {ScaleToFit};
table = Table({img, Paragraph('Entry 2')});
table.Border = 'solid';
table.Width = '2in';
table.entry(1,1).Style = {Height('lin'), Width('lin')};
table.entry(1,2).Border = 'solid';
append(d,table);

close(d);
rptview(d.OutputPath);
```



## **See Also**

mreportgen.dom.Image | mreportgen.dom.Height | mreportgen.dom.Width

## **Topics**

“Create and Format Images” on page 13-88

## **Introduced in R2014b**

## mlreportgen.dom.Strike class

**Package:** mlreportgen.dom

Strike through text

### Description

Specifies whether to use a strikethrough line for a text object. Strike appears as a single, horizontal line drawn through the text.

### Construction

`strikeObj = Strike()` draws a single, horizontal line through text.

`strikeObj = Strike(type)` draws a line of the specified type through text.

### Input Arguments

#### **type** — Strike type

'single' (default) | 'none' | 'double'

Strike type, specified as:

- 'single' — Single horizontal line (default)
- 'none' — No strikethrough line
- 'double' — Double horizontal line

### Output Arguments

#### **strike** — Strike through text

mlreportgen.dom.Strike object

An mlreportgen.dom.Strike object representing strikethrough text.

### Properties

#### **Id** — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

#### **Tag** — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the

object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Value — Strike type**

'single' | 'none' | 'double'

Strike type, specified as one of these values:

- 'single' — Single horizontal line (default)
- 'none' — No strikethrough line
- 'double' — Double horizontal line

**See Also**

mlreportgen.dom.Underline

**Topics**

“Report Formatting Approaches” on page 13-17

## mlreportgen.dom.StyleRef class

**Package:** mlreportgen.dom

Placeholder for reference to content with specified style name or outline level

### Description

Create a placeholder for a reference to content that has a specified style name or outline level. This object applies to Word and PDF reports.

For a Microsoft Word document, you can append a `StyleRef` object to the header, footer, or in the body text. For PDF, you can append a `StyleRef` object only to the header or footer.

---

**Tip** Use `StyleRef` objects to create running headers and footers in your document. For example, you can use this object to add the title of the current chapter in the page header.

---

### Construction

`styleref = StyleRef()` creates a reference to the content of the paragraph nearest to this object whose `OutlineLevel` property of 1.

In the headers of Word output, the nearest paragraph is the first paragraph on the current page that has the specified outline level. If there is no such paragraph on the current page, the nearest paragraph is the first paragraph on pages before or after the current page that has the specified outline level.

In the footers of Word output, the nearest paragraph is the last paragraph on the current page that has the specified outline level. If there is no such paragraph on the current page, the nearest paragraph is the first paragraph on pages before or after the current page that has the specified outline level.

In page headers and footers in PDF output, the nearest paragraph is the first paragraph on the current page or on pages in the current page layout section before or after the current page.

`styleref = StyleRef(num)` creates a reference to the content of the paragraph nearest to this object whose `OutlineLevel` property has the specified level.

`styleref = StyleRef(styleName)` creates a reference to the content of the paragraph nearest to this object that has the specified style name.

### Input Arguments

**num — Level of heading object to reference**

positive integer

Level of the heading object to reference, specified as a positive integer.

**styleName — Name of style of object to reference**

character vector

Name of style of object to reference, specified as a character vector.

## Output Arguments

### **styleRef** — Referenced object

mlreportgen.dom.StyleRef object

Referenced object, returned as an mlreportgen.dom.StyleRef object.

## Properties

### **Children** — Children of document element

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

### **CustomAttributes** — Custom attributes of this element

array of mlreportgen.dom.CustomAttribute objects

Custom attributes of this element, specified as an array of mlreportgen.dom.CustomAttribute objects. Use custom attributes supported by the output format.

### **Id** — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **RefOutlineLevel** — Outline (heading) level of object to reference

positive integer

### **RefStyleName** — Style sheet style name of object to reference

character vector

Style sheet style to apply to the reference, specified as a character vector.

### **Parent** — Parent of document element

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

### **Style** — Format specification

array of format objects

Format objects that specify the format of a document element.

### **StyleName** — Style to apply from style sheet

character vector

Name of the style to apply from the style sheet, specified as a character vector.

### **Tag** — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Methods

Method	Purpose
clone	Copy this object.

## Examples

### Add Reference to Heading1 Content in a Footer

This example uses an outline level to specify the content of a running footer.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'pdf');
open(d);

% Create page footer
footer = PDFPageFooter('default');
d.CurrentPageLayout.PageFooters = footer;

% Define and the StyleRef object using default (first level heading)
% Append it to the footer
ref = StyleRef();
append(footer, ref);

% Create several pages
% The footer content changes based on the last Heading1 object
h = Heading1('My First Head');
p = Paragraph('The above heading appears in the footer because it is a level 1 head.');
```

```
append(d, h);
append(d, p);

h2 = Heading1('My Next Head');
h2.Style = {PageBreakBefore(true)};
p2 = Paragraph('Now the above heading appears in the footer.');
```

```
append(d, h2);
append(d, p2);

h3 = Heading1('My Third Head');
h3.Style = {PageBreakBefore(true)};
append(d, h3);
append(d, clone(p2));

p3 = Paragraph(['Because I have not added another Heading1 object '...
'since the last one, the heading from the previous page appears in the footer.']);
p3.Style = {PageBreakBefore(true)};
append(d, p3);
```

```
close(d);
rptview(d.OutputPath);
```

### Add Reference to Content from Named Style

This example shows how to specify a style name for the contents of the reference. This example creates two `StyleRef` objects: one that uses the default value (`Heading1` objects) and one that uses the content of a paragraph formatted with the `Subtitle` style name. You insert both objects in the footer so that the footer contains text in the form `[Most Recent Heading1 Name]: [Most Recent Subtitle Name]`.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'docx');
open(d);

% Create page footer
footer = DOCXPageFooter('default');
d.CurrentPageLayout.PageFooters = footer;

% Create two StyleRef objects. ref uses content of Heading1 objects;
% ref2 uses content of paragraphs that use Subtitle style name.
ref = StyleRef();
ref2 = StyleRef('Subtitle');

% Assemble the footer text
footpara = Paragraph();
footpara.WhiteSpace = 'preserve';
append(footpara, ref);
append(footpara, ': ');
append(footpara, ref2);
append(footer, footpara);

% Create Heading1 and Subtitle paragraphs
% Footers update based on most recent values
h = Heading1('My Document Title');
sub = Paragraph('Subtitle Text');
sub.StyleName = 'Subtitle';
p = Paragraph('Here''s some text. ');
append(d, h);
append(d, sub);
append(d, p);

sub2 = Paragraph('Another Subtitle');
sub2.StyleName = 'Subtitle';
sub2.Style = {PageBreakBefore(true)};
append(d, sub2);
append(d, clone(p));

close(d);
rptview(d.OutputPath);
```

### See Also

[mlreportgen.dom.Paragraph](#) | [mlreportgen.dom.PageRef](#) | [mlreportgen.dom.Heading1](#) | [mlreportgen.dom.Heading](#) | [mlreportgen.dom.DOCXPageHeader](#) |

m\reportgen.dom.DOCXPageFooter | m\reportgen.dom.PDFPageHeader |  
m\reportgen.dom.PDFPageFooter

**Topics**

“Report Formatting Approaches” on page 13-17

**Introduced in R2016a**



# mlreportgen.dom.Table class

**Package:** mlreportgen.dom mlreportgen.dom

Informal table

## Description

Use objects of the `mlreportgen.dom.Table` class to define an informal table, which is a table that has only a body. It does not have separate header or footer sections.

The `mlreportgen.dom.Table` class is a `handle` class.

## Class Attributes

<code>HandleCompatible</code>	<code>true</code>
<code>ConstructOnLoad</code>	<code>true</code>

## Creation

### Description

`tableObj = mlreportgen.dom.Table(nCols)` creates an empty table and sets the `NCol` property to `nCols`. Use this syntax as the starting point for creating a table.

`tableObj = mlreportgen.dom.Table(array)` creates a table whose content is specified by `array`. The constructor converts basic MATLAB types to corresponding DOM types. For example, it changes character vectors to `mlreportgen.dom.Text` objects.

`tableObj = mlreportgen.dom.Table(array, style)` creates a table and sets the `Style` property to `style`. The style specified by `style` must be defined in the template used to create the document to which this table is appended.

### Input Arguments

#### **array — Table body content**

two-dimensional numeric array | two-dimensional categorical array | two-dimensional cell array | two-dimensional string array

Table body content, specified as:

- A two-dimensional numeric array
- A two-dimensional categorical array
- A two-dimensional string array
- A two-dimensional cell array that contains arrays, data types, or objects of these type:
  - Character vector
  - String scalar
  - One-dimensional or two-dimensional cell array

- Items of data type `double`
- `mlreportgen.dom.Text` object
- `mlreportgen.dom.Paragraph` object
- `mlreportgen.dom.Image` object
- `mlreportgen.dom.Table` object
- `mlreportgen.dom.FormalTable` object
- `mlreportgen.dom.OrderedList` object
- `mlreportgen.dom.UnorderedList` object
- `mlreportgen.dom.ExternalLink` object
- `mlreportgen.dom.InternalLink` object
- `mlreportgen.dom.CharEntity` object
- `mlreportgen.dom.LineBreak` object

## Properties

### BackgroundColor — Background color

[ ] (default) | character vector | string scalar

Table background color, specified as a character vector or string scalar that consists of a CSS color name or a hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use `#` as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, `"#0000ff"` specifies blue.

Setting the `BackgroundColor` property adds a `mlreportgen.dom.BackgroundColor` format object to the `Style` property. Setting the `BackgroundColor` property to an empty value removes the object.

Example: `"blue"`

Example: `"#0000ff"`

#### Attributes:

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>
<code>NonCopyable</code>	<code>true</code>

### Border — Type of border to draw

`"solid"` | `"single"` | `"dashed"` | `"none"` | ...

Type of border to draw, specified as one of the values in the table.

Border Value	Description	Supported Output Types
<code>"dashed"</code>	Dashed line	All output types
<code>"dashdotstroked"</code>	Line with alternating diagonal dashes and dot	Word

<b>Border Value</b>	<b>Description</b>	<b>Supported Output Types</b>
"dashsmallgap"	Dashed line with a small gap between dashes	Word
"dotted"	Dotted line	All output types
"dotdash"	Line with alternating dots and dashes	Word
"dotdotdash"	Line with alternating double dots and a dash	Word
"double"	Double line	All output types
"doublewave"	Double wavy line	Word
"groove"	3-D effect grooved line	HTML and PDF
"hidden"	No line  The "none" border type also produces no line. However, conflicting borders are handled differently for "hidden" types than for "none" types. The "hidden" border type takes precedence over a conflicting border type. A conflicting border type takes precedence over the "none" type.	HTML and PDF
"inset"	3-D effect line	All output types
"none"	No line  The "hidden" border type also produces no line. However, conflicting borders are handled differently for "hidden" types than for "none" types. The "hidden" border type takes precedence over a conflicting border type. A conflicting border type takes precedence over the "none" type.	All output types
"outset"	3-D effect line	All output types
"ridge"	3-D effect ridged line	HTML and PDF
"single"	Single line	Word
"solid"	Single line	HTML and PDF
"thick"	Thick line	Word
"thickthinlargegap"	Dashed line with alternating thick and thin dashes with a large gap	Word

Border Value	Description	Supported Output Types
"thickthinmediumgap"	Dashed line with alternating thick and thin dashes with a medium gap	Word
"thickthinsmallgap"	Dashed line with alternating thick and thin dashes with a small gap	Word
"thinthicklargegap"	Dashed line with alternating thin and thick dashes with a large gap	Word
"thinthickmediumgap"	Dashed line with alternating thin and thick dashes with a medium gap	Word
"thinthicksmallgap"	Dashed line with alternating thin and thick dashes with a small gap	Word
"thinthickthinlargegap"	Dashed line with alternating thin and thick dashes with a large gap	Word
"thinthickthinmediumgap"	Dashed line with alternating thin and thick dashes with a medium gap	Word
"thinthickthinsmallgap"	Dashed line with alternating thin and thick dashes with a small gap	Word
"threedemboss"	Embossed effect line	Word
"threedengrave"	Engraved effect line	Word
"triple"	Triple line	Word
"wave"	Wavy line	Word

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**BorderCollapse – Whether to collapse borders of adjacent cells into single border (HTML only)**

"on" | "off"

A value of "on" collapses borders of adjacent cells into a single border. A value of "off" keeps borders of adjacent cells. This property applies only to HTML documents.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**BorderColor — Border color**

character vector | string scalar

Table border color, specified as a character vector or string scalar that consists of a CSS color name or a hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, "#0000ff" specifies blue.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**BorderWidth — Table border width**

character vector | string scalar

Table border width, specified as a string scalar or character vector that consists of a number followed by an abbreviation of a unit of measurement. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Children — Children of this DOM API object**

array of DOM API objects

Children of this DOM API object, specified as an array of DOM API objects.

**Attributes:**

GetAccess	public
SetAccess	private
NonCopyable	true

**ColSep — Style of line separating columns**

character vector | string scalar

Style of the line that separates the columns of a table, specified as a character vector or string scalar.

See the description of the attribute `Border` for a description of the possible values.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>
<code>NonCopyable</code>	<code>true</code>

**ColSepColor — Color of line separating columns**

character vector | string scalar

Color of line separating columns, specified as a character vector or string scalar that consists of a CSS color name or a hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use `#` as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, `"#0000ff"` specifies blue.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>
<code>NonCopyable</code>	<code>true</code>

**ColSepWidth — Width of line separating table columns**

character vector | string scalar

Width of the line separating table columns, specified as a string scalar or character vector that consists of a number followed by an abbreviation of a unit of measurement. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>
<code>NonCopyable</code>	<code>true</code>

**ColSpecGroups — Properties of group of columns in table**

array of `mlreportgen.dom.TableColSpecGroup` objects

Width, alignment, and other properties of a group of columns, specified as an array of `mlreportgen.dom.TableColSpecGroup` objects. The first object applies to the first group of columns, the second object to the second group, and so on. Specify the number of columns belonging to each group by using the `Span` property of the `TableColSpecGroup` object. For example, if the first object has a span of 2, it applies to the first two columns. If the second group has a span of 3, it applies to the next three columns, and so on.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**CustomAttributes — Custom attributes for the table**

array of mlreportgen.dom.CustomAttribute objects

Custom attributes for the table, specified as an array of mlreportgen.dom.CustomAttribute objects. The custom attributes must be supported by the output type of the document to which this document element is appended.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**FlowDirection — Text flow direction**

"ltr" | "rtl"

Direction for text to flow, specified as one of these values:

- "ltr" — Text flows from left to right
- "rtl" — Text flows from right to left

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**HALign — Horizontal alignment of this table**

"center" | "left" | "right"

Horizontal alignment of this table, specified as one of these values:

- "center"
- "left"
- "right"

---

**Note** To prevent the overflow of large tables in PDF output, set the `Width` property.

---

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Id — ID for this DOM API object**

character vector | string scalar

ID for this DOM API object, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**NColumns — Number of columns in this table**

integer

Number of table columns, specified as an integer. To specify the number of columns, create a table by using the syntax `mlreportgen.dom.Table(nColumns)`. Otherwise, the `Table` constructor determines the number of columns from the table content. If you add rows to a table or entries to a row and the number of columns changes, the value of the `NColumns` property updates. If the rows do not all have the same number of entries, the row with the largest number of table entries determines the number of columns in the table.

**Attributes:**

GetAccess	public
SetAccess	private
NonCopyable	true

**NRows — Number of rows in this table**

integer

Number of rows in this table, specified as an integer. The `Table` constructor determines the number of rows from the table content. If you add rows to the table, the value of the `NRows` property updates.

**Attributes:**

GetAccess	public
SetAccess	private
NonCopyable	true

**OuterLeftMargin — Left margin of document element**

character vector | string scalar

Left margin of the table, specified as a string scalar or character vector that consists of a number followed by an abbreviation of a unit of measurement. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points



**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Parent — Parent of this DOM API object**

DOM API object

Parent of this DOM API object, specified as a DOM API object.

**Attributes:**

GetAccess	public
SetAccess	private
NonCopyable	true

**RowSep — Style of lines separating rows**

character vector | string scalar

Style of the lines that separate the rows of a table, specified as a character vector or string scalar.

See the description of the `Border` property for a description of the possible values.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**RowSepColor — Color of lines separating table rows**

character vector | string scalar

Color of lines separating table rows, specified as a character vector or string scalar that consists of a CSS color name or a hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, "#0000ff" specifies blue.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**RowSepWidth — Width of lines separating table rows**

character vector | string scalar

Width of the lines that separates the table rows, specified as a string scalar or character vector that consists of a number followed by an abbreviation of a unit of measurement. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches

- mm — millimeters
- pc — picas
- pt — points

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Style — Format for table**

array of format objects

Formats that define the style of this table, specified as a cell array of DOM format objects. The formats override the corresponding formats defined by the stylesheet style specified by the `StyleName` property.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**StyleName — Style in document or document part style sheet**

character vector | string scalar

Name of a style specified in the style sheet of the document or document part to which this table is appended, specified as a character vector or string scalar.

The style specified by the `StyleName` property must be defined in the stylesheet of the document or document part to which this table is appended. The specified style defines the appearance of the table in the output document, except for formats that are specified by the `Style` property of this `Table` object. The format objects specified by the `Style` property override formats defined by the stylesheet.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**TableEntriesHAlign — Horizontal alignment of table entries**

"center" (default) | "left" | "right"

Horizontal alignment of table entries, specified as one of these values:

- "center"
- "left"
- "right"

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**TableEntriesVAlign — Vertical alignment of table cell content**

"top" | "middle" | "bottom"

Vertical alignment of table cell content, specified as one of these values:

- "top"
- "middle"
- "bottom"

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**TableEntriesInnerMargin — Inner margin for table entries**

character vector | string scalar

Inner margin for the table entries, specified as a string scalar or character vector that consists of a number followed by an abbreviation of a unit of measurement. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**TableEntriesStyle — Style to use for table entries**

cell array of format objects

Style to use for the table entries, specified as a cell array of format objects.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Tag — Tag for this DOM API object**

character vector | string scalar

Tag for this DOM API object, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Attributes:**

```
GetAccess           public
SetAccess          public
NonCopyable        true
```

**Width — Table width**

character vector | string scalar

Percentage of the page width, such as "100%", or a number of units of measurement, specified as a string scalar or character vector that consists of a number followed by an abbreviation of a unit of measurement. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

For Word report, the width is calculated as a percentage of the page width minus the margins.

**Attributes:**

```
GetAccess           public
SetAccess          public
NonCopyable        true
```

## Methods

### Public Methods

Method	Purpose
entry	Access the entries of the Table
append	Append DOM object to Table
row	Access the rows of the Table
clone	Copy this Table object. Use the clone method for a Table object in the same way that you use the clone method for a Paragraph object.

## Examples

## Create a Table

In this example, you create a table by using a `Table` object and specify its properties.

Import the DOM API package so that you do not have to use long, fully qualified names.

```
import mlreportgen.dom.*;
```

Create a `Document` object and open it.

```
d = Document("myreport", "html");
open(d);
```

Create a table object and specify its content by using a 5-by-5 MATLAB array. The number array is converted to a table of DOM objects of the same dimension.

```
t = Table(magic(5));
```

Specify the row height.

```
t.Style = {RowHeight("1in")};
```

Specify the width and style of row and column separators.

```
t.Border = "solid";
t.BorderWidth = "1px";
t.ColSep = "solid";
t.ColSepWidth = "1";
t.RowSep = "solid";
t.RowSepWidth = "1";
```

Set the style for the table entries.

```
t.TableEntriesStyle = [t.TableEntriesStyle {FontFamily("Arial"),Width("1in"),Color("red"),Bold}]
```

Set the horizontal and vertical alignment for the text in the table.

```
t.TableEntriesHAlign = "center";
t.TableEntriesVAlign = "middle";
```

Add the table to the document.

```
append(d,t);
```

Close the document and view the report.

```
close(d);
rptview(d);
```

## See Also

`mlreportgen.dom.FormalTable` | `mlreportgen.dom.TableBody` |  
`mlreportgen.dom.TableRow` | `mlreportgen.dom.TableEntry` |  
`mlreportgen.dom.TableHeaderEntry` | `mlreportgen.dom.MATLABTable` |  
`mlreportgen.dom.TableColSpecGroup`

## Topics

“Create Informal Tables” on page 13-64

“Format Tables” on page 13-72

**Introduced in R2015b**

# mlreportgen.dom.TableBody class

**Package:** mlreportgen.dom

Body of formal table

## Description

Specifies the body of a formal table

## Properties

### Children — Children of document element

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

### ColSep — Style of line separating columns

character vector

The style of the line separating the columns of a table or table section (header, body, footer), as specified by an mlreportgen.dom.ColSep object.

See the description of the Border property for a description of the possible values.

### ColSepColor — Color of line separating columns

character vector

Color of line separating columns, specified as either:

- Name of a color. The name must be a CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- A hexadecimal RGB (truecolor) value as #RRGGBB. For example, #0000ff is a shade of blue.

### ColSepWidth — Width of line separating table columns

character vector

Width of the line separating table columns, in the form valueUnits. Use one of these abbreviations for the Units:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

For example, for a column separator of 3 points, set the ColSepWidth property to '3pt'.

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**RowSep — Style of lines separating rows**

character vector

The style of a line separating the rows of a table or table section (header, body, or footer).

See the description of the `Border` property for a description of the possible values.

**RowSepColor — Color of lines separating table rows**

character vector

Color of lines separating table rows, specified as one of these values:

- The name of a color. See the `mlreportGen.dom.Color` class reference page for a list of supported colors.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Style — Format specification**

array of format objects

Format objects that specify the format of a document element.

**StyleName — Table body style name**

character vector

The style specified by `styleName` must be defined in the template used to create the document element to which this table body is appended.

**TableEntriesHAlign — Horizontal alignment of table entries**

'center' (default) | 'left' | 'right'

Horizontal alignment of table entries, specified as one of these values:

- 'center'
- 'left'
- 'right'

Data Types: char

**TableEntriesVAlign — Vertical alignment of table cell content**

'top' | 'middle' | 'bottom'

Vertical alignment of table cell content, specified as one of these values:

- 'top'
- 'middle'
- 'bottom'



**TableEntriesInnerMargin — Inner margin for table entries**

character vector

The inner margin is the margin between table cell content and the cell borders in the form `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**TableEntriesStyle — Style to use for table entries**

cell array

Cell array of format objects that specify the format for table entries.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Methods**

Method	Purpose
<code>append</code>	Appends content to a table body.
<code>entry</code>	Get a table entry.
<code>row</code>	Create a table row.

**See Also**

`mlreportgen.dom.Table` | `mlreportgen.dom.FormalTable` | `mlreportgen.dom.TableRow` | `mlreportgen.dom.TableFooter` | `mlreportgen.dom.TableHeader` | `mlreportgen.dom.TableEntry` | `mlreportgen.dom.TableHeaderEntry`

**Topics**

“Create Informal Tables” on page 13-64

## mlreportgen.dom.TableColSpec class

**Package:** mlreportgen.dom

Formatting for one or more adjacent table columns

### Description

Define the formatting for one or more adjacent table columns. Use a `TableColSpec` object to override formats specified by a `TableColSpecGroup` object.

### Construction

`colSpecObj = TableColSpec()` creates a column specification having a span of 1.

### Output Arguments

**colSpecObj** — Formatting for one or more adjacent table columns

mlreportgen.dom.TableColSpec object

Formatting for one or more adjacent table columns, represented by an mlreportgen.dom.TableColSpec object.

### Properties

**CustomAttributes** — Custom attributes of document element

array of mlreportgen.dom.CustomAttribute objects

Custom attributes of this document element, specified as an array of mlreportgen.dom.CustomAttribute objects. The custom attributes must be supported by the output format.

**Id** — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Span** — Number of adjacent table columns to which this document element applies

number of adjacent table columns

If this property is not specified (its value is []), the value is assumed to be 1.

Data Types: double

**Style** — Style of adjacent table columns

array of format objects

Format objects that specify the style of the columns governed by this specification.

**Tag** — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### **See Also**

mlreportgen.dom.TableColSpecGroup | mlreportgen.dom.ResizeToFitContents |  
mlreportgen.dom.Table | mlreportgen.dom.FormalTable

### **Topics**

“Create Informal Tables” on page 13-64

## mlreportgen.dom.TableColSpecGroup class

**Package:** mlreportgen.dom

Define style for group of table columns

### Description

Use an object of the mlreportgen.dom.TableColSpecGroup class to define the style for a group of adjacent table columns. To override the style specified by a TableColSpecGroup object for one or more adjacent columns of the group, use an mlreportgen.dom.TableColSpec object.

The mlreportgen.dom.TableColSpecGroup class is a handle class.

### Class Attributes

ConstructOnLoad	true
HandleCompatible	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

colSpecGroupObj = mlreportgen.dom.TableColSpecGroup() creates a column style specification that spans an entire table.

## Properties

### Span — Number of adjacent table columns to which the column formatting applies

[ ] (default) | integer

Number of adjacent table columns to which the column formatting applies, specified as an integer. If the value of the Span property is empty ([ ]), the formatting applies to all of the columns in the table.

### Style — Column formatting

array of format objects

Column formatting, specified as an array of format objects. Format objects that do not apply to a TableColSpecGroup object are ignored.

### ColSpecs — Styles for column subgroups

[ ] (default) | array of mlreportgen.dom.TableColSpec objects

Styles for column subgroups, specified as an array of mlreportgen.dom.TableColSpec objects. The first mlreportgen.dom.TableColSpec object applies to the first subgroup of the group of columns specified by this TableColSpecGroup object. The second TableColSpec object applies to the next subgroup, and so on. For example, suppose that this TableColSpecGroup object applies to the first five columns of the table and that the first TableColSpec object in the ColSpecs property

has a span of two. The first `TableColSpec` object applies to the first and second columns of the table.

The style specified by a `TableColSpec` object in the `ColSpecs` property overrides the style specified by the `Style` property of this `TableColSpecGroup` object.

**StyleName — This property is ignored**

[] (default) | character vector | string scalar

This property is ignored.

**CustomAttributes — Custom attributes of this document element**

[] (default) | array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this document element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. The custom attributes must be supported by the report output format.

**Parent — This property is ignored**

[] (default)

This property is ignored and is read-only.

**Children — This property is ignored**

1-by-0 cell array (default)

This property is ignored and is read-only.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

## Examples

### Specify Color of Columns in a Table

Make the first column of a table green and the remaining columns red. Use a `TableColSpecGroup` object to specify the color of all of the table columns and use a `TableColSpec` object to override the color for the first column.

```
import mlreportgen.dom.*
d = Document('mydoc', 'docx');
```

```
append(d, 'Table');

grps(1) = TableColSpecGroup;
grps(1).Style = {Color('red')};
grps(1).Span = 5;
specs(1) = TableColSpec;
specs(1).Style = {Color('green')};
grps(1).ColSpecs = specs;
table = Table(magic(5));
table.ColSpecGroups = grps;

append(d, table);

close(d);
rptview(d);
```

## See Also

[mlreportgen.dom.TableColSpec](#) | [mlreportgen.dom.ResizeToFitContents](#) |  
[mlreportgen.dom.Table](#) | [mlreportgen.dom.FormalTable](#) |  
[mlreportgen.dom.MATLABTable](#)

## Topics

“Format Tables” on page 13-72  
“Create Informal Tables” on page 13-64  
“Create Formal Tables” on page 13-66  
“Create Tables from MATLAB Tables” on page 13-68

## Introduced in R2014b

# mlreportgen.dom.TableEntry class

**Package:** mlreportgen.dom

Table entry

## Description

Specifies the content and style of a table entry.

---

**Tip** To specify formatting for all table entries in a table, use the `TableEntriesStyle` property of the `Table` or `FormalTable` object. For example, you can set border formatting.

```
import mlreportgen.dom.*
t = Table(magic(5));
t.TableEntriesStyle = {Border('solid','black','1')};
```

Properties you set for a `TableEntry` object take precedence over `TableEntriesStyle` format objects.

---

The `mlreportgen.dom.TableEntry` class is a handle class.

## Creation

### Description

`entryObj = TableEntry()` creates an empty table entry.

`entryObj = TableEntry(text)` creates a table entry using the specified text. The constructor creates a text object and appends it to the table entry. In Microsoft Word and PDF output, text in a table entry is wrapped in a paragraph because Word and PDF do not permit unwrapped text in table entries. In HTML output, the text is not wrapped in a paragraph.

`entryObj = TableEntry(text,styleName)` creates a table entry containing specified text using the specified style.

`entryObj = TableEntry(domObj)` creates a table entry containing `domObj`, where `domObj` is a DOM object such as an `mlreportgen.dom.Paragraph` object.

### Input Arguments

#### **text** — Table entry text

character vector

Text for the table entry, specified as a character vector.

#### **textObj** — Text object containing the text for the table entry

mlreportgen.dom.Text object

Text for the table entry, specified as an `mlreportgen.dom.Text` object.

**styleName — Style for the table**

character vector

The style specified by `styleName` must be defined in the template of the document to which this table is appended.

**domObject — Objects to include in table**

DOM object

Objects to include in the table, specified as a DOM object. The valid DOM objects are:

- `mlreportgen.dom.Paragraph`
- `mlreportgen.dom.Text` (CharEntity included)
- `mlreportgen.dom.Image`
- `mlreportgen.dom.Table`
- `mlreportgen.dom.OrderedList`
- `mlreportgen.dom.UnorderedList`
- `mlreportgen.dom.CustomElement`

**Properties****Border — Type of border to draw**

'solid' | 'single' | 'dashed' | 'none' | ...

Type of border to draw, specified as one of the values in the table.

Border Value	Description	Supported Output Types
'dashed'	Dashed line	All output types
'dashdotstroked'	Line with alternating diagonal dashes and dot	Word
'dashsmallgap'	Dashed line with a small gap between dashes	Word
'dotted'	Dotted line	All output types
'dotdash'	Line with alternating dots and dashes	Word
'dotdotdash'	Line with alternating double dots and a dash	Word
'double'	Double line	All output types
'doublewave'	Double wavy line	Word
'groove'	3-D effect grooved line	HTML and PDF



<b>Border Value</b>	<b>Description</b>	<b>Supported Output Types</b>
'hidden'	No line  The 'none' border type also produces no line. However, conflicting borders are handled differently for 'hidden' types than for 'none' types. The 'hidden' border type takes precedence over a conflicting border type. A conflicting border type takes precedence over the 'none' type.	HTML and PDF
'inset'	3-D effect line	All output types
'none'	No line  The 'hidden' border type also produces no line. However, conflicting borders are handled differently for 'hidden' types than for 'none' types. The 'hidden' border type takes precedence over a conflicting border type. A conflicting border type takes precedence over the 'none' type.	All output types
'outset'	3-D effect line	All output types
'ridge'	3-D effect ridged line	HTML and PDF
'single'	Single line	Word
'solid'	Single line	HTML and PDF
'thick'	Thick line	Word
'thickthinlargegap'	Dashed line with alternating thick and thin dashes with a large gap	Word
'thickthinmediumgap'	Dashed line with alternating thick and thin dashes with a medium gap	Word
'thickthinsmallgap'	Dashed line with alternating thick and thin dashes with a small gap	Word
'thinthicklargegap'	Dashed line with alternating thin and thick dashes with a medium gap	Word
'thinthickmediumgap'	Dashed line with alternating thin and thick dashes, with a medium gap	Word

Border Value	Description	Supported Output Types
'thinthicksmallgap'	Dashed line with alternating thin and thick dashes with a small gap	Word
'thinthickthinlargegap'	Dashed line with alternating thin and thick dashes with a large gap	Word
'thinthickthinmediumgap'	Dashed line with alternating thin and thick dashes with a medium gap	Word
'thinthickthinsmallgap'	Dashed line with alternating thin and thick dashes with a small gap	Word
'threedemboss'	Embossed effect line	Word
'threedengrave'	Engraved effect line	Word
'triple'	Triple line	Word
'wave'	Wavy line	Word

**BorderColor — Border color**

character vector

Border color, specified as either:

- Name of a color. The name must be a CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- A hexadecimal RGB (truecolor) value as #RRGGBB. For example, #0000ff is a shade of blue.

**BorderWidth — Table border width**

character vector

Table border width, specified in the form `valueUnits`, where `Units` is an abbreviation for the units. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Children — Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**ColSpan — Number of table columns spanned by table entry**

double

Number of table columns spanned by the table entry, specified as a double.

Data Types: `double`

### **CustomAttributes — Custom attributes for document element**

array of `mlreportgen.doc.CustomAttribute` objects

The custom attributes must be supported by the output type of the document to which this document element is appended.

### **Hyphenation — Type of hyphenation**

`false` | `true` | `' - '` | `' '`

Type of hyphenation, specified as one of these values:

- `true` — Enables hyphenation and uses `' - '` for the hyphenation character.
- `' - '` — Enables hyphenation and uses `' - '` for the hyphenation character.
- `' '` — Enables hyphenation and uses a space (`' '`) for the hyphenation character
- `false` — Disables hyphenation.

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **InnerMargin — Inner margin (padding) around entry**

character vector

Inner margin in the form `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points
- `px` — pixels

### **Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

### **RowSpan — Number of table rows spanned by table entry**

`double`

Number of table rows spanned by the table entry, specified as a double.

Data Types: `double`

**Style – Format for table**

array of format objects

Array of format objects (such as **Bold** objects) that specify the format for this table.

This property overrides corresponding formats defined by the style sheet style specified by the `StyleName` property.

**StyleName – Style in document or document part style sheet**

character vector

Name of a style specified in the style sheet of the document or document part to which this table is appended

The style that specifies the appearance of this table in the output document, for formats not specified by `Style` property.

**Tag – Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**VALign – Vertical alignment table cell content**

'top' | 'bottom' | 'middle'

Vertical alignment table cell content, specified as one of these values:

- 'top'
- 'bottom'
- 'middle'

**Methods****Public Methods**

Use `TableEntry.append` and `TableEntry.clone` methods the same way you use `Paragraph.append` and `Paragraph.clone`.

Method	Purpose
<code>append</code>	Append text, paragraphs, images, tables, and other elements to this table entry.  In Microsoft Word and PDF output, text in a table entry is wrapped in a paragraph because Word and PDF do not permit unwrapped text in table entries. In HTML output, the text is not wrapped in a paragraph.

Method	Purpose
clone	Clone this table entry.

### See Also

mlreportgen.dom.Table | mlreportgen.dom.FormalTable | mlreportgen.dom.TableBody | mlreportgen.dom.TableRow | mlreportgen.dom.TableFooter | mlreportgen.dom.TableHeader | mlreportgen.dom.TableHeaderEntry | mlreportgen.dom.TableHeaderEntry | mlreportgen.dom.TextOrientation

### Topics

“Create Informal Tables” on page 13-64

### Introduced in R2013b

## mlreportgen.dom.TableFooter class

**Package:** mlreportgen.dom

Formal table footer

### Description

Specifies the content and format of a formal table footer.

### Properties

#### Children — Children of document element

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

#### ColSep — Style of line separating columns

character vector

The style of the line separating the columns of a table or table section (header, body, footer), as specified by an mlreportgen.dom.ColSep object.

See the description of the Border property for a description of the possible values.

#### ColSepColor — Color of line separating columns

character vector

Color of line separating columns, specified as either:

- Name of a color. The name must be a CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- A hexadecimal RGB (truecolor) value as #RRGGBB. For example, #0000ff is a shade of blue.

#### ColSepWidth — Width of line separating table columns

character vector

Width of the line separating table columns, in the form valueUnits. Use one of these abbreviations for the Units:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

For example, for a column separator of 3 points, set the ColSepWidth property to '3pt'.

#### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

### **RowSep — Style of lines separating rows**

character vector

The style of a line separating the rows of a table or table section (header, body, or footer).

See the description of the `Border` property for a description of the possible values.

### **RowSepColor — Color of lines separating table rows**

character vector

Color of lines separating table rows, specified as one of these values:

- The name of a color. See the `mlreportGen.dom.Color` class reference page for a list of supported colors.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

### **RowSepWidth — Width of lines separating table rows**

character vector

Width of lines separating table rows in the form `valueUnits`, where `Units` is an abbreviation for the units. Use one of these abbreviations:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

### **Style — Format for the table footer**

array of format objects

Array of format objects (such as `Bold` objects) that specify the format for the table footer.

This property overrides corresponding formats defined by the style sheet style specified by the `StyleName` property.

### **StyleName — Style in the document or document part style sheet**

character vector

Name of a style specified in the style sheet of the document or document part to which the table footer is appended

Data Types: `char`

**TableEntriesHAlign — Horizontal alignment of table entries**

'center' (default) | 'left' | 'right'

Horizontal alignment of table entries, specified as 'center', 'left', or 'right'.

Data Types: char

**TableEntriesInnerMargin — Inner margin for table entries**

character vector

The inner margin is the margin between table cell content and the cell borders in the form `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**TableEntriesStyle — Style to use for table entries**

cell array

Cell array of format objects that specify the format for table entries.

**TableEntriesVAlign — Vertical alignment of table cell content**

'top' | 'middle' | 'bottom'

Vertical alignment of table cell content, specified as one of these values:

- 'top'
- 'middle'
- 'bottom'

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Methods**

Method	Purpose
append	Appends a row of table entries to this table footer.
entry	Get a footer entry



Method	Purpose
row	Get a footer row

**See Also**

mlreportgen.dom.TableHeader | mlreportgen.dom.FormalTable |  
mlreportgen.dom.TableHeaderEntry | mlreportgen.dom.TableBody |  
mlreportgen.dom.TableRow | mlreportgen.dom.TableEntry

**Topics**

“Create Informal Tables” on page 13-64

## mlreportgen.dom.TableHeader class

**Package:** mlreportgen.dom

Table header

### Description

Table header for labeling columns.

### Properties

#### Children — Children of document element

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

#### ColSep — Style of line separating columns

character vector

The style of the line separating the columns of a table or table section (header, body, footer), as specified by an mlreportgen.dom.ColSep object.

See the description of the Border property for a description of the possible values.

#### ColSepColor — Color of line separating columns

character vector

Color of line separating columns, specified as either:

- Name of a color. The name must be a CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- A hexadecimal RGB (truecolor) value as #RRGGBB. For example, #0000ff is a shade of blue.

#### ColSepWidth — Width of line separating table columns

character vector

Width of the line separating table columns, in the form valueUnits. Use one of these abbreviations for the Units:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

For example, for a column separator of 3 points, set the ColSepWidth property to '3pt'.

#### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **RowSep — Style of lines separating rows**

character vector

The style of a line separating the rows of a table or table section (header, body, or footer).

See the description of the `Border` property for a description of the possible values.

### **RowSepColor — Color of lines separating table rows**

character vector

Color of lines separating table rows, specified as one of these values:

- The name of a color. See the `mlreportGen.dom.Color` class reference page for a list of supported colors.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

### **RowSepWidth — Width of lines separating table rows**

character vector

Width of lines separating table rows in the form `valueUnits`, where `Units` is an abbreviation for the units. Use one of these abbreviations:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

### **Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

### **Style — Format for the table header**

array of format objects

Array of format objects (such as `Bold` objects) that specify the format for the table header.

This property overrides corresponding formats defined by the style sheet style specified by the `StyleName` property.

### **StyleName — Style in the document or document part style sheet**

character vector

Name of a style specified in the style sheet of the document or document part to which the table header is appended

The style that specifies the appearance of the table header in the output document, for formats not specified by `Style` property.

**TableEntriesHAlign — Horizontal alignment of table entries**

'center' (default) | 'left' | 'right'

Horizontal alignment of table entries, specified as 'center', 'left', or 'right'.

Data Types: char

**TableEntriesInnerMargin — Inner margin for table entries**

character vector

The inner margin is the margin between table cell content and the cell borders in the form `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**TableEntriesStyle — Style to use for table entries**

cell array

Cell array of format objects that specify the format for table entries.

**TableEntriesVAlign — Vertical alignment of table cell content**

'top' | 'middle' | 'bottom'

Vertical alignment of table cell content, specified as one of these values:

- 'top'
- 'middle'
- 'bottom'

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Methods**

Method	Purpose
append	Appends a row of table entries to this table header.
entry	Get a header entry.

Method	Purpose
row	Get a header row.

**See Also**

mlreportgen.dom.TableFooter | mlreportgen.dom.FormalTable |  
mlreportgen.dom.TableHeaderEntry | mlreportgen.dom.TableBody |  
mlreportgen.dom.TableRow | mlreportgen.dom.TableEntry

**Topics**

“Create Informal Tables” on page 13-64

## mlreportgen.dom.TableHeaderEntry class

**Package:** mlreportgen.dom

Entry in table header

### Description

Specifies a table header entry.

This class is intended primarily to support HTML table creation. It is rendered in an HTML document as a `th` (table header cell) element. Use this element so you do not need to format a table header entry explicitly. `TableHeaderEntry` objects rely on the browser to render the header entry appropriately. For Microsoft Word and PDF output, `TableHeaderEntry` behaves the same as `TableEntry`.

### Construction

`entryObj = TableHeaderEntry()` creates an empty table header entry.

### Output Arguments

#### **entryObj** — Table header entry

mlreportgen.dom.TableHeaderEntry object

Table header entry, represented by an mlreportgen.dom.TableHeaderEntry object.

### Properties

#### **Border** — Type of border to draw

'solid' | 'single' | 'dashed' | 'none' | ...

Type of border to draw, specified as one of the values in the table.

Border Value	Description	Supported Output Types
'dashed'	Dashed line	All output types
'dashdotstroked'	Line with alternating diagonal dashes and dot	Word
'dashsmallgap'	Dashed line with a small gap between dashes	Word
'dotted'	Dotted line	All output types
'dotdash'	Line with alternating dots and dashes	Word
'dotdotdash'	Line with alternating double dots and a dash	Word
'double'	Double line	All output types

<b>Border Value</b>	<b>Description</b>	<b>Supported Output Types</b>
'doublewave'	Double wavy line	Word
'groove'	3-D effect grooved line	HTML and PDF
'hidden'	No line  The 'none' border type also produces no line. However, conflicting borders are handled differently for 'hidden' types than for 'none' types. The 'hidden' border type takes precedence over a conflicting border type. A conflicting border type takes precedence over the 'none' type.	HTML and PDF
'inset'	3-D effect line	All output types
'none'	No line  The 'hidden' border type also produces no line. However, conflicting borders are handled differently for 'hidden' types than for 'none' types. The 'hidden' border type takes precedence over a conflicting border type. A conflicting border type takes precedence over the 'none' type.	All output types
'outset'	3-D effect line	All output types
'ridge'	3-D effect ridged line	HTML and PDF
'single'	Single line	Word
'solid'	Single line	HTML and PDF
'thick'	Thick line	Word
'thickthinlargegap'	Dashed line with alternating thick and thin dashes with a large gap	Word
'thickthinmediumgap'	Dashed line with alternating thick and thin dashes with a medium gap	Word
'thickthinsmallgap'	Dashed line with alternating thick and thin dashes with a small gap	Word
'thinthicklargegap'	Dashed line with alternating thin and thick dashes with a medium gap	Word

Border Value	Description	Supported Output Types
'thinthickmediumgap'	Dashed line with alternating thin and thick dashes, with a medium gap	Word
'thinthicksmallgap'	Dashed line with alternating thin and thick dashes with a small gap	Word
'thinthickthinlargegap'	Dashed line with alternating thin and thick dashes with a large gap	Word
'thinthickthinmediumgap'	Dashed line with alternating thin and thick dashes with a medium gap	Word
'thinthickthinsmallgap'	Dashed line with alternating thin and thick dashes with a small gap	Word
'threedemboss'	Embossed effect line	Word
'threedengrave'	Engraved effect line	Word
'triple'	Triple line	Word
'wave'	Wavy line	Word

**BorderColor – Border color**

character vector

Border color, specified as either:

- Name of a color. The name must be a CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- A hexadecimal RGB (truecolor) value as #RRGGBB. For example, #0000ff is a shade of blue.

**BorderWidth – Table border width**

character vector

Table border width, specified in the form valueUnits, where Units is an abbreviation for the units. Valid abbreviations are:

- px – pixels (default)
- cm – centimeters
- in – inches
- mm – millimeters
- pc – picas
- pt – points

**Children – Children of document element**

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.



**ColSpan — Number of table columns spanned by table entry**

double

Number of table columns spanned by the table entry, specified as a double.

Data Types: double

**CustomAttributes — Custom attributes for document element**array of `mlreportgen.doc.CustomAttribute` objects

The custom attributes must be supported by the output type of the document to which this document element is appended.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Hyphenation — Type of hyphenation**

false | true | '-' | ' '

Type of hyphenation, specified as one of these values:

- `true` — Enables hyphenation and uses '-' for the hyphenation character.
- '-' — Enables hyphenation and uses '-' for the hyphenation character.
- ' ' — Enables hyphenation and uses a space (' ') for the hyphenation character
- `false` — Disables hyphenation.

**InnerMargin — Inner margin (padding) around entry**

character vector

Inner margin in the form `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**RowSpan — Number of table rows spanned by table entry**

double

Number of table rows spanned by the table entry, specified as a double.

Data Types: double

**Style — Format for table**

array of format objects

Array of format objects (such as `Bold` objects) that specify the format for this table.

This property overrides corresponding formats defined by the style sheet style specified by the `StyleName` property.

**StyleName — Style in document or document part style sheet**

character vector

Name of a style specified in the style sheet of the document or document part to which this table is appended

The style that specifies the appearance of this table in the output document, for formats not specified by `Style` property.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**VAlign — Vertical alignment table cell content**

'top' | 'bottom' | 'middle'

Vertical alignment table cell content, specified as one of these values:

- 'top'
- 'bottom'
- 'middle'

**Methods**

Use `TableHeaderEntry.append` and `TableHeaderEntry.clone` methods the same way you use `Paragraph.append` and `Paragraph.clone`.

Method	Purpose
<code>append</code>	Append text, paragraphs, images, tables, and other elements to this entry
<code>clone</code>	Clone this table header entry

**See Also**

`mlreportgen.dom.TableHeader` | `mlreportgen.dom.TableFooter` |  
`mlreportgen.dom.FormalTable` | `mlreportgen.dom.TableBody` |  
`mlreportgen.dom.TableRow`

**Topics**

“Create Informal Tables” on page 13-64

## mlreportgen.dom.TableRow class

**Package:** mlreportgen.dom

Table row

### Description

Use objects of the mlreportgen.dom.TableRow class to create a table row.

The mlreportgen.dom.TableRow class is a handle class.

### Class Attributes

ConstructOnLoad	true
HandleCompatible	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

tableRowObj = TableRow() creates an empty table row.

## Properties

### Height — Height of table row

[] (default) | character vector | string scalar

Height of this table row, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '0.5in' specifies one-half inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

If the Style property of this table row includes an mlreportgen.dom.RowHeight format object, the Height property is set to the height specified by the format object.

If you set the Height property to a height value, a RowHeight object with the specified height is created and added to the Style property of the row, or is used to replace an existing RowHeight object in the Style property. The Type of the new RowHeight object is 'exact'. This Type value causes Microsoft Word to generate a row of the specified height and truncate content that does not

fit. HTML and PDF viewers create a row of at least the specified height and adjust the row height to accommodate the content.

---

**Note** If you add an `mlreportgen.dom.Height` object to the `Style` property, it is converted to an `mlreportgen.dom.RowHeight` object with the `Type` set to `'atleast'`. This `Type` value causes HTML and PDF viewers and Microsoft Word to create a row of at least the specified height and adjust the row height to accommodate the content.

---

Example: `'0.5in'`

### Entries — Table entries in row

array of `mlreportgen.dom.TableEntry` objects

Table entries in this row, specified as an array of `mlreportgen.dom.TableEntry` objects. Use this property to access the table entries in this row. For example, this code accesses element 2 in row 2:

```
t = Table({'e11', 'e12'; 'e21', 'e22'});
elem22 = t.row(2).Entries(2);
```

You can also access element 2 in row 2 by using the `entry` method of the `mlreportgen.dom.Table` class. For example:

```
t = Table({'e11', 'e12'; 'e21', 'e22'});
elem22 = entry(t,2,2);
```

Once you access the `TableEntry` object that corresponds to a table entry, you can format the entry by setting properties of the object. See “Format a Table Entry” on page 12-487.

This property is read-only.

### NEntries — Number of entries in row

0 (default) | integer

Number of table entries in this row, specified as an integer. This property is read-only.

### StyleName — Name of style for formatting table row

[] (default) | character vector | string scalar

Name of the style for formatting this table row, specified as a character vector or string scalar.

The style specified by the `StyleName` property must be defined in the stylesheet of the document or document part to which this table row is appended. The specified style defines the appearance of the table row in the output document, except for formats that are specified by the `Style` property of this table row. The format objects specified by the `Style` property override the formats defined in the style.

The `StyleName` property is ignored for Word output.

### Style — Formats that define table row style

cell array of DOM format objects

Formats that define the style of this table row, specified as a cell array of DOM format objects. The formats override the corresponding formats defined by the stylesheet style specified by the `StyleName` property.

You can specify the row height by adding an `mlreportgen.dom.RowHeight` or an `mlreportgen.dom.Height` object to the `Style` property. An `mlreportgen.dom.Height` object is converted to an `mlreportgen.dom.RowHeight` object with the `type` set to `'atleast'`.

### CustomAttributes — Custom attributes of document element

array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this document element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. The custom attributes must be supported by the output format.

### Parent — Parent of document element

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

### Children — Children of document element

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

## Methods

### Public Methods

Method	Purpose
<code>append</code>	Append content to table row
<code>clone</code>	Clone this table row. Use the <code>clone</code> method of the <code>mlreportgen.dom.TableRow</code> class in the same way as you use the <code>clone</code> method of the <code>mlreportgen.dom.Paragraph</code> class.

## Examples

## Add Content to an Empty Table

To add content to an empty table, append table entries to table rows and then append the table rows to the table. This example creates this two-by-two table:

```
e11 e12
e21 e22
```

Create a document and then create a table that has two columns.

```
import mlreportgen.dom.*
```

```
d = Document();
t = Table(2);
```

Create two table rows.

```
tr1 = TableRow();
tr2 = TableRow();
```

Create table entries that contain the content and append the table entries to the rows.

```
append(tr1,TableEntry('e11'));
append(tr1,TableEntry('e12'));
append(tr2,TableEntry('e21'));
append(tr2,TableEntry('e22'));
```

Append the table rows to the table.

```
append(t,tr1);
append(t,tr2);
```

Append the table to the document. Close and view the document.

```
append(d,t);
close(d);
rptview(d);
```

## Format a Table Entry

Use the `Entries` property of an `mlreportgen.dom.TableRow` object to access the `mlreportgen.dom.TableEntry` object that corresponds to the entry that you want to format. Format the entry by setting format properties of the `TableEntry` object or by adding format objects to the `Style` property of the object. This example changes the text color of the second entry of the second row to red.

```
import mlreportgen.dom.*
d = Document();
t = Table({'e11', 'e12'; 'e21', 'e22'});
t.row(2).Entries(2).Style = {Color('red')};
append(d,t);
close(d);
rptview(d);
```

In the resulting table, the text, e22, in the second entry of the second row is red.

```
e11 e12  
e21 e22
```

Alternatively, you can access a table entry by using the `entry` method of the `m1reportgen.dom.Table` object that contains the entry. In the previous example, replace:

```
t.row(2).Entries(2).Style = {Color('red')};
```

with:

```
elem = entry(t,2,2);  
elem.Style = {Color('red')};
```

### **See Also**

`m1reportgen.dom.TableEntry` | `m1reportgen.dom.Table` | `m1reportgen.dom.FormalTable`  
| `m1reportgen.dom.TableBody` | `m1reportgen.dom.TableHeader` |  
`m1reportgen.dom.TableFooter`

### **Topics**

“Create Informal Tables” on page 13-64

**Introduced in R2014b**



# mlreportgen.dom.Template class

**Package:** mlreportgen.dom

Create report template object

## Description

Create a report template object.

Use mlreportgen.dom.Template objects to create templates. For example, you can append DOM content, such as Text, Paragraph, or Image objects, and TemplateHole objects to a Template object to create a template containing fixed content with holes for generated content.

---

**Note** Word for Mac does not support creating holes for DOM API templates. If you need to create a Word template for generating Word documents on a Mac, you can create a template using the DOM API. Create a Template object and use mlreportgen.dom.TemplateHole to add holes. Alternatively, use Word on Windows to create your template and copy the template to your Mac.

---

## Construction

templateObj = Template() creates a template object based on the default HTML template. The resulting template is in the current folder and uses the name Untitled.htmxtx.

Append content and use a corresponding close command to generate the template file.

templateObj = Template(templatePath) creates a template object that outputs a template file at the specified location. The default template type if you do not specify an extension is HTML.

templateObj = Template(templatePath,type) creates the template of the specified type. If you specify an extension using templatePath, the types must match.

---

**Tip** Use a variable for the type argument to simplify your code. See “Create a Template and Add Content” on page 12-493 for an example.

---

templateObj = Template(templatePath,type,sourceTemplatePath) creates a template based on the template sourceTemplatePath.

## Input Arguments

### templatePath — Template to create

character vector

Full path of template you want to create, specified as a character vector. You can use an extension to create a template of the corresponding type:

- .htmxtx for HTML ( default)
- .docx for Word

- `.html` for single-file HTML
- `.pdf` for PDF

**type — Type of template**`'html' (default) | 'docx' | 'html-file'`

Type of template, specified as one of these values:

- `'html'` — HTML template as a zipped or unzipped folder containing the HTML document text, image, style sheet, and JavaScript files
- `'docx'` — Word template
- `'html-file'` — HTML template consisting of a single file that contains the text, style sheets, JavaScript, and images for the report
- `.pdf` — PDF template

If you use an extension with the `templatePath` input argument, the `type` argument must match.

**sourceTemplatePath — Basis for new template**`character vector`

Template that is the basis for the new template, specified as a character vector. The source template type must match the `type` argument.

**Output Arguments****templateObj — Template object to create**`mlreportgen.dom.Template object`

Template to create, returned as an `mlreportgen.dom.Template` object.

**Properties****Children — Children of document element**`array of DOM objects`

Children of this document element, specified as an array of DOM objects. This property is read-only.

**CurrentHoleId — ID of current hole in document**`character vector`

This read-only property is the hole ID of the current hole in this document.

**CurrentHoleType — Type of current hole**`'Inline' | 'Block'`

Type of the current template hole, specified as `'Inline'` or `'Block'`.

- An inline hole is for document elements that a paragraph element can contain: `Text`, `Image`, `LinkTarget`, `ExternalLink`, `InternalLink`, `CharEntity`, `AutoNumber`.
- A block hole can contain a `Paragraph`, `Table`, `OrderedList`, `UnorderedList`, `DocumentPart`, or `Group`.

**CurrentPageLayout — Current page layout of this document**

mlreportgen.dom.DOCXPageLayout object | mlreportgen.dom.PDFPageLayout object

This property applies to Word and PDF documents. For Word documents, the value is a `DOCXPageLayout` object that specifies the current page layout. For PDF documents, the value is a `PDFPageLayout` object if the document currently specifies a page layout. For HTML documents, the value is always `[]`.

**ForceOverwrite — Option to overwrite existing output file**`[]` (default) | logical value

Set this property to `true` to overwrite an existing output file of the same name. If this property is `false` and a writable file of the same name exists, attempting to close (i.e., write) this template causes an error. If the existing file is read-only, closing this document causes an error regardless of the setting of this property.

Data Types: logical

**HTMLHeadExt — Custom content for HTML header**

character vector

Custom content for HTML header, specified as a character vector.

Data Types: char

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**OpenStatus — Open status of document element**

unopened (default) | open | closed

This read-only property lists the open status of this document element.

**OutputPath — Path of output file or folder**

character vector

Path of the output file or folder, specified as a character vector. If you do not specify the file extension, the DOM adds an extension based on the document format. You can set this property only before opening the document.

For unzipped output packaging, the path specifies the folder for the output files. The default is the current folder.

**PackageType — Packaging for files generated**

'zipped' (default) | 'unzipped' | 'both' | 'single-file'

Packaging for output files generated, specified as one of these values:

- 'zipped' — Applies only to Word, PDF, and multifile HTML output.
- 'unzipped' — Applies only to Word, PDF, and multifile HTML output.
- 'both' — Applies only to Word, PDF, and multifile HTML output.

- `'single-file'` — Creates the report as a single file. This value appears if you set the document's `Type` property to `'html-file'`. You cannot set or change this value yourself.

For zipped packaging, the document output is a zip file located at the location specified by the `OutputPath` property. The zip file has the extension that matches the document type: `docx` for Word output, `pdfx` for PDF output, or `htmlx` for HTML output. For example, if the document type is `docx` and `OutputPath` is `s:\docs\MyDoc`, the output is packaged in a zip file named `s:\docs\MyDoc.docx`.

For unzipped packaging, the document output is stored in a folder having the root file name of the `OutputPath` property. For example, if the `OutputPath` is `s:\docs\MyDoc`, the output folder is `s:\docs\MyDoc`.

If you set `PackageType` to both, generating the report produces zipped and unzipped output.

Data Types: `char`

### **StreamOutput — Option to stream output to disk**

`false` (default) | logical value

By default, document elements are stored in memory until the document is closed. Set this property to `true` to write document elements to disk as the elements are appended to the document.

Data Types: `logical`

### **Tag — Tag for this document**

session-unique tag (default) | character vector

Tag that identifies this document. The tag has the form `CLASS:ID`, where `CLASS` is the document class and `ID` is the value of the `Id` property of the object.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during document generation.

### **TemplatePath — Path of the template to use**

character vector

The full path to the HTML or Word template to use, specified as a character vector.

### **TitleBarText — Title for HTML browser title bar**

character vector

For HTML documents, this property specifies the text that appears in the title bar of the browser used to display this document. Word and PDF documents ignore this property.

Set this property before opening the document for output.

### **Type — Type of output**

`'html'` (default) | `'docx'` | `'html-file'` | `'pdf'`

Type of output, specified as one of these values:

- `'html'` — HTML output as a zipped or unzipped folder containing the HTML document text, image, style sheet, and JavaScript files
- `'docx'` — Word output

- 'html-file' — HTML output consisting of a single file that contains the text, style sheets, JavaScript, and images for the report
- 'pdf' — PDF output

If you specify a template using the `TemplatePath` property, the template must be consistent with the `Type` property.

## Methods

Use the `Template` methods the same way you use the corresponding `Document` methods.

Method	Purpose
<code>append</code>	Append document element to the document.
<code>close</code>	Close this document. You cannot close a document if it has not been opened or was previously closed.
<code>mlreportgen.dom.Document.createTemplate</code>	Create default template.
<code>fill</code>	Fill document hole.
<code>mlreportgen.dom.Document.getCoreProperties</code>	Get core properties of document.
<code>mlreportgen.dom.Document.getImageDirectory</code>	Get image directory for the document.
<code>mlreportgen.dom.Document.getImagePrefix</code>	Get generated image name prefix for the document.
<code>getMainPartPath</code>	Get relative path of main part of output document.
<code>mlreportgen.dom.Document.getOPCMainPart</code>	Get full path of main part of output document.
<code>moveToNextHole</code>	Move to next template hole.
<code>open</code>	Open this document. You cannot open a document if it was previously opened or closed.
<code>package</code>	Append file to OPC package of document.
<code>mlreportgen.dom.Document.setCoreProperties</code>	Set core properties of document element.

## Examples

### Create a Template and Add Content

This example creates a template with a hole for the title and a hole for the author. You can change the value of the `type` variable to create a template of one of the other types.

```
import mlreportgen.dom.*;

type = 'docx';
```

```
% Create a template object
t = Template('mytemplate',type);

% Add a title hole to the template and apply the Title style
hole = append(t,TemplateHole('TITLE'));
hole.Description = ('Title Description');
hole.DefaultHoleStyleName = 'Title';

% Add a paragraph with boilerplate text and apply the Subtitle format
% Position the paragraph and preserve white space in the text
p = Paragraph('Author: ');
p.StyleName = 'Subtitle';
p.Style = {OuterMargin('0','0','lin','lin')};
p.WhiteSpace = 'preserve';

% Append an inline hole to paragraph
hole = append(p,TemplateHole('AUTHOR'));
append(t,p);

close(t);
```

This example uses the template to fill the holes.

```
% Create a document TitleAuthor that uses the template mytemplate.
rpt = Document('TitleAuthor',type,'mytemplate');
open(rpt);

% Create a loop to cycle through the holes.
% Append content to each hole.
while(~strcmp(rpt.CurrentHoleId,'#end#'))
    switch(rpt.CurrentHoleId)
        case 'TITLE'
            append(rpt,Paragraph('This Is My Title'));
        case 'AUTHOR'
            append(rpt,'My Name');
    end

    moveToNextHole(rpt);
end

% Generate and view the report.
close(rpt);
rptview(rpt.OutputPath)
```

## See Also

[mlreportgen.dom.TemplateHole](#) | [mlreportgen.dom.Document.createTemplate](#) | [rptview](#)

## Topics

“Create a Microsoft Word Template” on page 13-119

“Create an HTML or PDF Template” on page 13-130

“Use Style Sheet Styles” on page 13-19

# mlreportgen.dom.TemplateHole class

**Package:** mlreportgen.dom

Hole to append to template

## Description

Hole to append to a document template.

You can append a template hole to these kinds of DOM objects:

- Paragraph
- TableEntry
- Group
- Template

## Construction

`templateHoleObj = TemplateHole()` creates a hole with empty properties.

`templateHoleObj = TemplateHole(id)` creates a hole having the specified id.

`templateHoleObj = TemplateHole(id,description)` creates a hole having the specified id and description.

## Input Arguments

### **id — ID for template hole**

character vector

The ID for the template hole, specified as a character vector.

### **description — Description for template hole**

character vector

Description for the template hole, specified as a character vector. The value of this argument becomes the content of the hole in the template to which it is assigned to allow you to determine the purpose of the hole when viewing the template in the corresponding application. The description is replaced by appended hole content in a report generated from the template.

## Output Arguments

### **templateHoleObj — Hole to append to template**

mlreportgen.dom.TemplateHole object

Hole to append to template, represented by an mlreportgen.dom.TemplateHole object.

## Properties

### DefaultHoleStyleName — Name of default style for hole content

character vector

Name of default style for hole content. This style name is assigned to hole content that does not specify a style name. For example, suppose you append a `Text` object to this hole and the `Text` object does not specify a style name. Then the value of this property is assigned to the text object as its style name. This property allows a template to specify the appearance of appended content.

### Description — Description of this hole

character vector

Description for the template hole, specified as a character vector. The value of this property becomes the content of the hole in the template to which it is assigned to allow you to determine the purpose of the hole when viewing the template in the corresponding application. The description is replaced by appended hole content in a report generated from the template.

### HoleId — ID of this hole

character vector

ID of this template hole.

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Methods

Method	Purpose
<code>clone</code> Use <code>TemplateHole.clone</code> in a similar way to how you use <code>Paragraph.clone</code> .	Clone this hole object.

## See Also

`mlreportgen.dom.Template` | `mlreportgen.dom.Document.createTemplate` | `rptview`

## Topics

“Create a Microsoft Word Template” on page 13-119



“Create an HTML or PDF Template” on page 13-130  
“Use Style Sheet Styles” on page 13-19

## mlreportgen.dom.Text class

**Package:** mlreportgen.dom mlreportgen.dom mlreportgen.dom

Text object

### Description

Use an object of the mlreportgen.dom.Text class to include text in a document.

The mlreportgen.dom.Text class is a handle class.

### Class Attributes

HandleCompatible	true
ConstructOnLoad	true

## Creation

### Description

textObj = mlreportgen.dom.Text() creates an empty Text object.

textObj = mlreportgen.dom.Text(text) creates a Text object that contains the specified text and sets the Content property to text.

textObj = mlreportgen.dom.Text(text, styleName) also specifies the style and sets the StyleName property to styleName. You must define the style in the style sheet in the template of the document to which you append this Text object.

## Properties

### Content — Text contained by this document element

character vector | string scalar

Text contained by this document element, specified as a character vector.

### Attributes:

GetAccess	public
SetAccess	public
NonCopyable	true

### Bold — Whether to use bold for text

[] (default) | true or 1 | false or 0

Whether to use bold for text, specified as a numeric or logical 1 (true) or 0 (false). To make text bold, set Bold to true or 1.

Setting the Bold property adds a corresponding mlreportgen.dom.Bold format object to the Style property. Setting the Bold property to an empty value removes the object.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Italic — Whether to use italic for text**

[ ] (default) | true or 1 | false or 0

Whether to use italic for text, specified as a numeric or logical 1 (true) or 0 (false). To make text italic, set `Italic` to true or 1.

Setting the `Italic` property adds a corresponding `mlreportgen.dom.Italic` format object to the `Style` property. Setting the `Italic` property to an empty value removes the object.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Color — Text color**

[ ] (default) | character vector | string scalar

Text color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form `#RRGGBB`.

Setting the `Color` property adds a corresponding `mlreportgen.dom.Color` format object to the `Style` property. Setting the `Color` property to an empty value removes the object.

Example: "blue"

Example: "#0000ff"

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**BackgroundColor — Background color**

[ ] (default) | character vector | string scalar

Background color, specified as a character vector or string scalar that consists of a CSS color name (see <https://www.w3.org/wiki/CSS/Properties/color/keywords>) or a hexadecimal RGB (truecolor) value in the form `#RRGGBB`.

Setting the `BackgroundColor` property adds a corresponding `mlreportgen.dom.BackgroundColor` format object to the `Style` property. Setting the `BackgroundColor` property to an empty value removes the object.

Example: "blue"

Example: "#0000ff"

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Underline — Type of underline**

[ ] (default) | "single" | "none" | ...

Type of underline, specified as one of the values in this table.

Underline value	Description	Supported Output Types
"single"	Single line	All
"words"	Underline all characters except spaces	Word
"double"	Double line	Word
"thick"	Thick line	Word
"dotted"	Dotted line	Word
"dottedHeavy"	Thick dotted line	Word
"dash"	Dashed line	Word
"dashedHeavy"	Line with heavy dashes	Word
"dashLong"	Line with long dashes	Word
"dashLongHeavy"	Line with heavy long dashes	Word
"dotDash"	Dot-dash line	Word
"dashDotHeavy"	Heavy dash-dot line	Word
"dotDotDash"	Dot-dot-dash line	Word
"dashDotDotHeavy"	Line with heavy dashes and two dots between the dashes	Word
"wave"	Wavy line	Word
"wavyHeavy"	Heavy wavy	Word
"wavyDouble"	Double wavy line	Word
"none"	No underline	All

Setting the Underline property adds a corresponding `mlreportgen.dom.Underline` format object to the Style property. Setting the Underline property to an empty value removes the object.

To specify both the color and the type of the underline, do not set the Underline property. Instead, set the Style property to include an `mlreportgen.dom.Underline` format object that specifies the underline type and color.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**WhiteSpace — White space and line breaks in text**

[ ] (default) | "normal" | "nowrap" | ...

Type of white space and line breaks to use in text, specified as one of the values in the table. The `WhiteSpace` property does not apply when the `Type` property of the `Document` object is `PDF`.

Value	Description	Supported Output Types
"normal"	Does not preserve white space and line breaks.	Word and HTML
"nowrap"	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
"preserve"	Preserves white space. Text wraps when necessary and on line breaks. Acts like the <code>&lt;pre-wrap&gt;</code> setting in HTML.	Word and HTML
"pre"	Preserves white space. Text wraps only on line breaks. Acts like the <code>&lt;pre&gt;</code> setting in HTML.	HTML
"pre-line"	Sequences of white space collapse into a single white space. Text wraps.	HTML
"pre-wrap"	Preserves white space. Text wraps when necessary and on line breaks	HTML

If you want to view HTML output in the MATLAB browser and you want to preserve white space and wrap text only on line breaks, use the `preserve` setting instead of the `pre` setting.

Setting the `WhiteSpace` property adds a corresponding `WhiteSpace` format object to `Style` property. Removing the `WhiteSpace` property setting removes the `WhiteSpace` object.

#### Attributes:

```
GetAccess                public
SetAccess                 public
NonCopyable              true
```

#### FontFamilyName — Name of font family for text

[ ] | character vector | string scalar

Name of a font family to use for text, specified as a character vector or string scalar.

Setting the `FontFamilyName` property adds a corresponding `mlreportgen.dom.FontFamily` format object to the `Style` property. Setting the `FontFamilyName` property is the same as setting the `FontName` property of an `mlreportgen.dom.FontFamily` object. Setting the `FontFamilyName` property to an empty value removes the object.

To specify substitutions for the font, do not specify the font family by setting this property. Instead, create and add an `mlreportgen.dom.FontFamily` object to the `Style` property.

Example: "Courier New"

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**FontSize — Font size for text**

[] (default) | character vector | string scalar

Font size to use for text, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '12pt' specifies twelve points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

Setting the `FontSize` property adds a corresponding `mlreportgen.dom.FontSize` format object to the `Style` property. Setting the `FontSize` property to an empty value removes the object.

Example: "12pt"

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Strike — Text strikethrough**

[] (default) | "none" | "single" | "double"

Text strikethrough, specified as one of these values:

- "none" — No strikethrough
- "single" — Single line
- "double" — Double line (for Word reports only)

Setting the `Strike` property adds a corresponding `mlreportgen.dom.Strike` format object to the `Style` property for this document element. Setting the `Strike` property to an empty value removes the object.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**StyleName — Style for text**

character vector | string scalar

You must define the style in the style-sheet in the template of the document to which you append this Text object.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Style — Text formatting**

cell array of DOM format objects

Text formatting, specified as a cell array of DOM format objects.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**CustomAttributes — Custom attributes of document element**

array of mlreportgen.dom.CustomAttribute objects

Custom attributes of this document element, specified as an array of mlreportgen.dom.CustomAttribute objects. The custom attributes must be supported by the output format of the Document object to which this Text object is appended..

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Parent — Parent of this DOM API object**

DOM API object

Parent of this DOM API object, specified as a DOM API object.

**Attributes:**

GetAccess	public
SetAccess	private
NonCopyable	true

**Children — Children of this DOM API object**

array of DOM API objects

Children of this DOM API object, specified as an array of DOM API objects.

**Attributes:**

GetAccess	public
SetAccess	private
NonCopyable	true

**Tag — Tag for this DOM API object**

character vector | string scalar

Tag for this DOM API object, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Attributes:**

```
GetAccess          public
SetAccess          public
NonCopyable       true
```

**Id — ID for this DOM API object**

character vector | string scalar

ID for this DOM API object, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Attributes:**

```
GetAccess          public
SetAccess          public
NonCopyable       true
```

## Methods

### Public Methods

Method	Purpose
<code>append</code>	Append a custom element to this text object. Use the <code>append</code> method for a <code>Text</code> object in the same way that you use the <code>append</code> method for a <code>Paragraph</code> object.
<code>clone</code>	Copy this text object. Use the <code>clone</code> method for a <code>Text</code> object in the same way that you use the <code>clone</code> method for a <code>Paragraph</code> object.

## Examples

### Format Paragraph Text by Using Separate Text Objects

In this example, you format a paragraph by using two separate `mlreportgen.dom.Text` objects so that each piece of text can have its own style. The first `Text` object underlines and bolds the text. The second `Text` object uses the default style.

Import the DOM API package so that you do not have to use long, fully qualified names.

```
import mlreportgen.dom.*
```

Create a `Document` object.

```
d = Document("mydoc", "html");
```



Create a Paragraph object.

```
para = Paragraph();
```

Create a Text object for the text that you want to bold and underline. Append the Text object to the Paragraph object,

```
paraTitle = Text("Results:");  
paraTitle.Bold = true;  
paraTitle.Underline = "single";  
append(para,paraTitle);
```

Create a Text object for the rest of the text in the paragraph and append the Text object to the Paragraph object.

```
paraContent = Text(" Study 1");  
append(para,paraContent);  
append(d,para);
```

Close and view the report.

```
close(d);  
rptview(d);
```

Here is the generated paragraph:

**Results: Study 1**

## See Also

mlreportgen.dom.Paragraph | mlreportgen.dom.CharEntity |  
mlreportgen.dom.CustomText

## Topics

“Add Content to a Report” on page 13-10

“Report Formatting Approaches” on page 13-17

## Introduced in R2014b

## mlreportgen.dom.TextOrientation class

**Package:** mlreportgen.dom

Orientation of text in a table entry

### Description

Specifies the orientation for text in a table entry.

### Construction

`textOrientationObj = TextOrientation()` causes text to flow from left to right and for the first column to be on the left side of a table.

`textOrientationObj = TextOrientation(orientation)` causes text in a table entry to display with the specified orientation.

### Input Arguments

#### **orientation — Text orientation**

'horizontal' | 'down' | 'up'

Text orientation, specified as one of these values:

- 'horizontal' — text is horizontal in the text entry
- 'down' — text is vertical, with the first character at the top
- 'up' — text is vertical, with the first character at the bottom

### Output Arguments

#### **textOrientationObj — Text orientation**

mlreportgen.dom.TextOrientation object

Text orientation, returned as an mlreportgen.dom.TextOrientation object.

### Properties

#### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

#### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the

object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Value — Text orientation

'horizontal' | 'down' | 'up'

Text orientation, specified as one of these values:

- 'horizontal' — text is horizontal in the text entry
- 'down' — text is vertical, with the first character at the top
- 'up' — text is vertical, with the first character at the bottom

### Width — Table entry rotated text width

string (default)

Table entry rotated text width, specified as a string. This property applies only to PDF output. The width string is specified in the form `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- px — pixels
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

Example: 10px

Data Types: string

## Examples

### Vertical Text in a Table Entry

```
import mlreportgen.dom.*
d = Document('mydoc1', 'docx');

HeadStyle = {Bold, OuterMargin('0in')};
ch1 = Paragraph('Col1');
ch1.Style = HeadStyle;
ch2 = Paragraph('Col2');
ch2.Style = HeadStyle;

t = Table({ch1, ch2; 'entry1', 'entry2'});
EntryStyle = {TextOrientation('down'), VAlign('middle')};
t.entry(1,1).Style = EntryStyle;
t.entry(1,2).Style = EntryStyle;
t.row(1).Style = {RowHeight('24pt', 'atleast')};
append(d, t);
```

```
close(d);  
rptview(d.OutputPath);
```

### **See Also**

`mlreportgen.dom.TableEntry`

### **Topics**

“Create Informal Tables” on page 13-64

# mlreportgen.dom.TOC class

**Package:** mlreportgen.dom

Create placeholder for generating table of contents

## Description

Create a placeholder for a document table of contents. When a generated Word document opens, Word replaces the placeholder with a TOC that it generates. Similarly, when a generated HTML document opens in an HTML browser, the browser replaces the placeholder with a TOC that it generates. For PDF, the DOM API replaces the placeholder with a TOC that it generates when outputting the document.

In all cases, the TOC entries consist of the content of paragraphs using the number of heading levels that you specify. For PDF and Word, the TOC placeholder also specifies a leader that fills the space between the content and the page number in the TOC entry.

## Construction

`toc = TOC()` generates a three-level TOC that uses a dot leader.

`toc = TOC(levels)` uses the specified number of heading levels.

`toc = TOC(levels, leader)` uses the specified leader.

## Input Arguments

### **levels** — Number of heading levels to use in TOC

positive integer

Number of heading levels to use in TOC, specified as a positive integer.

### **leader** — Type of leader to use between title and page number

' .' (default) | ' ' (space)

Type of leader to use between title and page number, specified as ' .' (i.e., period or dot) or ' ' (a space).

## Output Arguments

### **toc** — Table of contents

mlreportgen.dom.TOC object

Table of contents, returned as an mlreportgen.dom.TOC object.

## Properties

### **Children** — Children of document element

array of DOM objects

Children of this document element, specified as an array of DOM objects. This property is read-only.

**CustomAttributes — Custom attributes of this element**

array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. Use custom attributes supported by the output format.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**LeaderPattern — Leader type for TOC**

' .' (default) | ' ' (space)

Type of leader to use between title and page number, specified as ' .' (i.e., period or dot) or ' ' (a space).

**NumberOfLevels — Number of heading levels to use in TOC**

positive integer

Number of heading levels to use in the TOC, specified as a positive integer.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**Style — Format specification**

array of format objects

Format objects that specify the format of a document element.

**StyleName — Style to apply from style sheet**

character vector

Name of the style to apply from the style sheet, specified as a character vector.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Examples**

## Insert a Table of Contents into a Document

This example adds a table of contents to a document using a TOC object. This document contains three levels of heads—|Heading1|, Heading2, and Heading3. Because the TOC object specifies only two heading levels, Heading3 is not included in the TOC. The leader is a space.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'pdf');
open(d);

title = append(d, Paragraph('My TOC Document'));
title.Bold = true;
title.FontSize = '28pt';

toc = append(d, TOC(2, ' '));
toc.Style = {PageBreakBefore(true)};

h1 = append(d, Heading1('Chapter 1'));
h1.Style = {PageBreakBefore(true)};
p1 = append(d, Paragraph('Hello World'));

h2 = append(d, Heading2('Section 1.1'));
h2.Style = {PageBreakBefore(true)};
p2 = append(d, Paragraph('Another page'));

h3 = append(d, Heading3('My Subsection 1.1.a'));
p3 = append(d, Paragraph('My Level 3 Heading Text'));

close(d);
rptview(d.OutputPath);
```

## See Also

mlreportgen.dom.Heading1 | mlreportgen.dom.PageBreakBefore |  
mlreportgen.dom.Paragraph

## Topics

“Report Formatting Approaches” on page 13-17

## Introduced in R2016a

## mlreportgen.dom.Underline class

**Package:** mlreportgen.dom

Draw line under text

### Description

Draw line under text

### Construction

`underline = Underline()` draws a single line under text.

`underline = Underline(type)` draws a line of the specified type under the text.

`underline = Underline(type,color)` draws a line of the specified type and color under the text. The color parameter must be an `mlreportgen.dom.Color` object.

### Input Arguments

#### **type** — Style of underline

character vector

Style of the underline, specified as one of these values. HTML and PDF support only 'single'.

Value	Description
'single'	Single underline (only style supported for HTML and PDF)
'double'	Double underline
'words'	Words only underlined (not spaces)
'thick'	Thick underline
'dotted'	Dotted underline
'dottedHeavy'	Thick, dotted underline
'dash'	Dashed underline
'dashedHeavy'	Thick, dashed underline
'dashLong'	Long, dashed underline
'dashLongHeavy'	Thick, long, dashed underline
'dotDash'	Dot dash underline
'dotDotDash'	Dash dot dot underline
'dashDotDotHeavy'	Thick dash dot dot underline
'dashDotHeavy'	Thick dash dot underline
'none'	No underline
'wave'	Wavy underline
'wavyDouble'	Two wavy underlines



Value	Description
'wavyHeavy'	Thick wavy underline

**color — Color of underline**

mlreportgen.dom.Color object

Color of the underline, specified by an mlreportgen.dom.Color object.

**Output Arguments****underline — Line under text**

mlreportgen.dom.Underline object

Underline, represented by an mlreportgen.dom.Underline object.

**Properties****Type — Underline style**

character vector

Underline style. See the description of the type input argument for the constructor.

Data Types: char

**Color — Color of underline**

mlreportgen.dom.Color object

Color of the underline, specified by an mlreportgen.dom.Color object.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**See Also**

mlreportgen.dom.Text

**Topics**

“Report Formatting Approaches” on page 13-17

## mlreportgen.dom.UnorderedList class

**Package:** mlreportgen.dom

Unordered (bulleted) list

### Description

Specifies an unordered (bulleted) list.

### Construction

`unorderedListObj = UnorderedList()` creates an empty unordered list.

`unorderedListObj = UnorderedList(items)` creates an unordered list of the specified list items.

### Input Arguments

#### **items — Content to include in unordered list**

array of doubles | array of character vectors | categorical array | cell array

Content to include in an unordered list, specified as an array of doubles, an array of character vectors, a categorical array, or a one-dimensional cell array.

The cell array can contain a combination of the following:

- A character vector
- A number
- A Boolean value
- One of the following DOM objects:
  - mlreportgen.dom.Text
  - mlreportgen.dom.Paragraph
  - mlreportgen.dom.ExternalLink
  - mlreportgen.dom.InternalLink
  - mlreportgen.dom.Table
  - mlreportgen.dom.Image
  - mlreportgen.dom.CustomElement
- Horizontal one-dimensional array (for a sublist)

To append an unordered list, use an `UnorderedList` DOM object instead of using the `items` argument.

### Output Arguments

#### **unorderedListObj — Unordered list**

mlreportgen.dom.UnorderedList object

An `mlreportgen.dom.UnorderedList` object representing an unordered list of the specified list items.

## Properties

### CustomAttributes — Custom attributes of document element

array of `mlreportgen.dom.CustomAttribute` objects

Custom attributes of this document element, specified as an array of `mlreportgen.dom.CustomAttribute` objects. The custom attributes must be supported by the output format.

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Style — Format specification

array of format objects

Format objects that specify the format of a document element.

### Stylename — List style name

character vector

The style specified by `Stylename` must be defined in the template used to create the document element to which you append this list.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Methods

Method	Purpose
<code>append</code> Use the <code>UnorderedList.append</code> method similar to how you use <code>OrderedList.append</code> .	Append items to this list.
<code>clone</code> Use the <code>UnorderedList.clone</code> method similar to how you use <code>Paragraph.clone</code> .	Copy the list.

## Examples

### Create an Unordered List

```
import mlreportgen.dom.*;
d = Document('mydoc');

ul = UnorderedList({Text('a'), 'b', 1, {'c', Paragraph('d')}});
append(d,ul);

close(d);
rptview('mydoc', 'html');
```

### See Also

`mlreportgen.dom.OrderedList` | `mlreportgen.dom.ListItem` |  
`mlreportgen.dom.ListStyleType`

### Topics

“Create and Format Lists” on page 13-51

# mlreportgen.dom.VAlign class

**Package:** mlreportgen.dom

Vertical alignment of document object

## Description

Specifies vertical alignment of objects.

## Construction

`vAlignObj = VAlign()` creates an alignment object having the value 'top'.

`vAlignObj = VAlign(value)` creates an alignment object having the specified value.

## Input Arguments

### **value — Specify vertical alignment**

'top' (default) | 'bottom' | 'middle'

Vertical alignment of a document element, specified as one of these values:

- 'top'
- 'middle'
- 'bottom'

## Output Arguments

### **vAlignObj — Vertical alignment of document object**

mlreportgen.dom.VAlign object

Vertical alignment of document object, represented by an mlreportgen.dom.VAlign object.

## Properties

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Value — Specify vertical alignment**

'top' (default) | 'bottom' | 'middle'

Vertical alignment of an object, specified as one of these values:

- 'top'
- 'middle'
- 'bottom'

**See Also**

`mlreportgen.dom.HAlign`

**Topics**

“Report Formatting Approaches” on page 13-17

# mlreportgen.dom.VerticalAlign class

**Package:** mlreportgen.dom

Vertical alignment of an inline document element

## Description

Specifies the vertical alignment of an inline document element, such as a text or image object.

The mlreportgen.dom.VerticalAlign class is a handle class.

## Class Attributes

ConstructOnLoad	true
HandleCompatible	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

mlreportgen.dom.VerticalAlign = VerticalAlign() specifies superscript alignment.

mlreportgen.dom.VerticalAlign = VerticalAlign(value) sets the Value property to the alignment specified by value.

## Properties

### Value — Vertical alignment

character vector

Vertical alignment of an inline document element, specified as one of these values:

- 'superscript' or 'super' — Aligns the element as superscript to the parent element.
- 'subscript' or 'sub' — Aligns the element as subscript to the parent element.
- 'baseline' — Aligns the baseline of the element with the baseline of the parent element.
- 'text-top' — Aligns the top of the element with the top of the “content area” on page 12-521 of the parent element. This value is not supported for Word documents.
- 'text-bottom' — Aligns the bottom of the element with the bottom of the “content area” on page 12-521 of the parent element. This value is not supported for Word documents.
- length — amount by which the element is raised or lowered from the baseline of the parent element. Format the length as valueUnits where value is the number of units and Units is the unit abbreviation. Use one of the abbreviations in the table.

Unit Name	Unit Abbreviation
pixels (not supported for Word documents)	px
centimeters	cm
inches	in
millimeters	mm
picas	pc
points	pt
percentage of the line height (not supported for Word documents)	%

See the vertical-align property in the CSS specification.

If you do not provide a `VerticalAlign` object, the alignment defaults to the baseline alignment. If you provide a `VerticalAlign` object, but do not specify the alignment, the alignment defaults to the superscript alignment.

Example: 'text-top'

Example: '0.25in'

Example: '50%'

#### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

#### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## **Examples**

### **Vertically Align Text as Superscript**

Align text as superscript to the text in the parent paragraph. Creating a `VerticalAlign` object without providing an input argument sets the alignment to superscript.

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);
p = Paragraph('e = mc');

t = Text('2');
```



```
t.Style = {VerticalAlign()};
append(p,t);
append(d,p);

close(d);
rptview('test',doctype);
```

### Align Image with Bottom of Text

Align the bottom of an image with the bottom of the surrounding text by specifying 'text-bottom'.

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('mydoc',doctype);

p = Paragraph('This image ');
p.FontSize = '20';
im = Image('image_to_align.png');
im.Style = {VerticalAlign('text-bottom')};
t = Text(' is aligned with the bottom of the surrounding text. ');
append(p,im);
append(p,t);
append(d,p);

close(d);
rptview('mydoc',doctype);
```

## More About

### content area

Area of an element that contains text and images, not including padding, borders, and margins.

The content area dimensions depend on the font and other factors and can vary by implementation. See content area height in the CSS specification.

### See Also

[mlreportgen.dom.Text](#) | [mlreportgen.dom.VAlign](#)

### Topics

“Report Formatting Approaches” on page 13-17

### External Websites

Cascading Style Sheets (CSS) Specification

### Introduced in R2014b

## mlreportgen.dom.WarningMessage class

**Package:** mlreportgen.dom

Warning message

### Description

Create a warning message with the specified text originating from the specified source object.

### Construction

`warningMsgObj = WarningMessage(text, source)` creates a warning message with the specified text originating from the specified source object.

#### Input Arguments

**text — Message text**

character vector

The text to display for the message.

**source — DOM object from which message originates**

a DOM object

The DOM object from which the message originates.

#### Output Arguments

**warningMsgObj — Warning message**

mlreportgen.dom.WarningMessage object

Warning message, represented by an mlreportgen.dom.WarningMessage object.

### Properties

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Source — Source DOM object from which message originates**

a DOM object

Source DOM object from which the message originates.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Text — Text of message

character vector

Message text, specified as a character vector.

## Methods

Use `WarningMessage` methods similar to how you use `ProgressMessage` methods.

Method	Purpose
<code>formatAsHTML</code>	Wrap message in HTML tags.
<code>formatAsText</code>	Format message as text.
<code>passesFilter</code>	Determine if message passes filter.

## Examples

### Create a Warning Message

```
import mlreportgen.dom.*;
d = Document('test','html');

dispatcher = MessageDispatcher.getTheDispatcher;
l = addlistener(dispatcher,'Message',...
    @(src, evtdata) disp(evtdata.Message.formatAsText));

open(d);
dispatch(dispatcher,WarningMessage('invalid chapter',d));
p = Paragraph('Chapter ');
p.Tag = 'chapter title';
p.Style = {CounterInc('chapter'),...
    CounterReset('table'),WhiteSpace('pre')};
append(p,AutoNumber('chapter'));
append(d,p);

close(d);
```

Delete the listener to avoid duplicate reporting of message objects during a MATLAB session.

```
delete(l);
```

### See Also

`dispatch`

### Topics

“Display Progress and Debugger Messages” on page 13-115

## mlreportgen.dom.Watermark class

**Package:** mlreportgen.dom

Add watermark to pages in sections of PDF reports

### Description

Creates a watermark object that you can add to a section of a PDF report. A watermark is an image that appears in the background of a page, such as the word `Draft` or `Confidential`. It runs behind the text on each page you apply it to. You can use any of these file types: `.bmp`, `.jpg`, `.png`, `.svg`, and `.tiff`.

### Construction

`wm = Watermark(image)` creates a `Watermark` object based on the specified image, and returns a `Watermark` object.

### Input Arguments

**image — Image to use as watermark**

path name

Image to use as the watermark, specified as the image path name. Use any of these file types:

- `.bmp`
- `.jpg`
- `.pdf` (for PDF output types only)
- `.png`
- `.svg`
- `.tiff`

### Properties

**Height — Watermark height**

character vector

Character vector in the form `valueUnits`. Use any of these values for units:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

Alternatively, You can specify the height using the `Watermark.Style` property. For example:

```
Watermark.Style = {Height('4in')};
```

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Path — Path of image file**

character vector

Path of image file, specified as a character vector.

**Style — Format specification**

array of format objects

Format objects that specify the format of a document element.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Width — Watermark width**

character vector

Watermark width, specified as a character vector in the form valueUnits. Use any of these values for units:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

Alternatively, you can specify the width using the Watermark.Style property. For example:

```
Watermark.Style = {Width('4in')};
```

## Examples

**Create and Insert Watermark**

This example shows how to create a watermark programmatically and then apply it to the current layout. Creating the watermark programmatically simplifies files management, because you do not need to store the image file and keep track of its location.

Using MATLAB® commands, create an image file programmatically. Using an SVG image file maintains the resolution as the image scales. After you write the image to a file, you can delete the figure.

```
wmname = 'wm';  
wmttype = 'svg';  
wmfilename = [wmname '.' wmttype];  
  
subplot('Position',[0, 0, 1, 1]);  
axis('off');  
text(0.25, 0.25,'Draft', ...  
     'Rotation', 45, ...  
     'Color', [0.85, 0.85, 0.85], ...  
     'FontSize',72);  
  
print(wmfilename, ['-d' wmttype]);  
delete(gcf);
```

Create the watermark object `wm` and apply it to the current page layout. After you generate the report, you can delete the image file specified by the variable `wmfilename`.

```
import mlreportgen.dom.*;  
  
d = Document('myreport','pdf');  
open(d);  
  
wm = Watermark(wmfilename);  
wm.Width = '12in';  
wm.Height = [];  
  
d.CurrentPageLayout.Watermark = wm;  
  
append(d,'Hello');  
append(d, PageBreak);  
append(d,'World');  
  
close(d);  
rptview(d.OutputPath);  
delete(wmfilename);
```

## See Also

`mlreportgen.dom.PDFPageLayout`

### Topics

“Report Formatting Approaches” on page 13-17

“Create Page Layout Sections” on page 13-144

### Introduced in R2016b

# mlreportgen.dom.WhiteSpace class

**Package:** mlreportgen.dom

White space type

## Description

Specifies behavior for white space and line breaks in text.

The mlreportgen.dom.WhiteSpace class is a handle class.

## Creation

### Description

`ws = WhiteSpace(option)` applies the specified white space option to white space in a Text or Paragraph object. For PDF, you can specify WhiteSpace only for a Paragraph object.

### Input Arguments

#### option — White space behavior

'normal' | 'nowrap' | 'pre' | 'pre-line' | 'preserve' | 'pre-wrap'

White space behavior, specified as one of these values.

---

**Note** Only 'preserve' and 'normal' affect Word output.

---

Value	Description
'preserve'	Preserves spaces and line breaks.
'normal' (default)	For HTML and PDF, removes leading and trailing spaces and collapses multiple spaces within text to a single space, ignoring line breaks.  For Word, removes leading and trailing spaces, ignoring line breaks.
'nowrap'	Sequences of white spaces collapse into a single white space. Text does not wrap to the next line. The text continues on the same line until a   tag is encountered.
'pre'	Preserves white space. Text wraps only on line breaks. Acts like the <pre> tag in HTML.
'pre-line'	Sequences of white spaces collapses into a single white space. Text wraps when necessary and on line breaks.
'pre-wrap'	Preserves white space. Text wraps when necessary and on line breaks.

## Properties

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### Tag — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### WhiteSpace — How to treat white space in text

[] (default) | 'normal' | 'nowrap' | ...

How to treat white space in text, specified as one of the values in this table.

Value	Description	Supported Output Types
'normal'	For HTML and PDF, this value removes spaces at the beginning and the end of text. Multiple spaces in the text collapse to a single space.  For Word, this value removes spaces at the beginning and end of text.	All
'nowrap'	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
'pre'	Preserves white space. Text wraps only on line breaks. Acts like the <pre> tag in HTML.	HTML and PDF
'pre-line'	Sequences of white space collapse into a single white space. Text wraps when necessary and on line breaks.	HTML and PDF
'pre-wrap'	Preserves white space. Text wraps when necessary and on line breaks.	HTML and PDF
'preserve'	Same as 'pre'.	All

Setting the WhiteSpace property adds a corresponding `mlreportgen.dom.WhiteSpace` format object to the Style property. Setting the WhiteSpace property to an empty value removes the object.



## Examples

### Preserve Space After Colon

This example shows the effect of using the 'preserve' option for each of the output formats. In HTML, multiple spaces collapse, but the trailing space is preserved. Preserving the trailing space is useful, for example, when you are creating a chapter title. Typically, you append an autonumber after the text 'Chapter: '. Using 'preserve' keeps the trailing space.

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);
open(d);

p = Paragraph('This paragraph has extra spaces   and one after the colon: ');
p.Style = {WhiteSpace('preserve')};

append(p,'XX');
append(d,p);

close(d);
rptview(d.OutputPath);
```

This example uses Word output. The multiple spaces do not collapse, and the trailing space is preserved. Try commenting out the WhiteSpace property. The multiple spaces are preserved, but the trailing space is removed.

```
import mlreportgen.dom.*;
doctype = 'docx';
d = Document('test',doctype);
open(d);

p = Paragraph('This paragraph has extra spaces   and one after the colon: ');
p.Style = {WhiteSpace('preserve')};

append(p,'XX');
append(d,p);

close(d);
rptview(d.OutputPath);
```

This example uses PDF output.

```
import mlreportgen.dom.*;
doctype = 'pdf';
d = Document('test',doctype);
open(d);

p = Paragraph('This paragraph has extra spaces   and one after the colon: ');
% p.Style = {WhiteSpace('preserve')};

append(p,'XX');
append(d,p);

close(d);
rptview(d.OutputPath);
```

## See Also

mlreportgen.dom.Text | mlreportgen.dom.Paragraph

## Topics

“Report Formatting Approaches” on page 13-17

**Introduced in R2014b**

## mlreportgen.dom.WidowOrphanControl class

**Package:** mlreportgen.dom

Widow and orphan handling

### Description

Specifies whether to prevent widows and orphans. This format applies only to Microsoft Word documents.

### Construction

`widowOrphanControlObj = WidowOrphanControl()` prevents a page break after the first line of a paragraph (orphan) or before the last line of a paragraph (widow).

`widowOrphanControlObj = WidowOrphanControl(tf)` prevents orphans and widows if `tf` is `true`.

### Input Arguments

#### **tf — Controls orphans and widows**

`true` (default) | `false` | `1` | `0`

A setting of `true` (or `1`) prevents a page break after the first line of a paragraph (orphan) or before the last line of a paragraph (widow). A setting of `false` (or `0`) allows a page break after the first line of a paragraph (orphan) or before the last line of a paragraph (widow).

Data Types: `logical`

### Output Arguments

#### **widowOrphanControlObj — Widow and orphan handling**

`mlreportgen.dom.WidowOrphanControl` object

Widow and orphan handling, represented by an `mlreportgen.dom.WidowOrphanControl` object.

### Properties

#### **Id — ID for this document element**

`character vector` | `string scalar`

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

#### **Tag — Tag for this document element**

`character vector` | `string scalar`

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the

object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Value — Control orphans and widows**

true (default) | false | 1 | 0

Possible values are:

- true or 1 — Prevents a page break after the first line of a paragraph (orphan) or before the last line of a paragraph (widow).
- false or 0 — Allows a page break after the first line of a paragraph (orphan) or before the last line of a paragraph (widow).

Data Types: logical

**See Also**

mlreportgen.dom.Paragraph

**Topics**

“Report Formatting Approaches” on page 13-17

## mlreportgen.dom.Width class

**Package:** mlreportgen.dom

Object width

### Description

Specifies the width of an object, such as an image or a table entry.

### Construction

`widthObj = Width()` creates a format object that specifies a width of 1 inch.

`widthObj = Width(value)` creates a width object having the specified width.

### Input Arguments

#### **value — Width of object**

character vector

Width of object, such as an image or a table entry, specified in the form `valueUnits`, where `Units` is an abbreviation for the units. These abbreviations are valid:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points
- `%` — percent

### Output Arguments

#### **widthObj — Object width**

mlreportgen.dom.Width object

Object width, represented by an `mlreportgen.dom.Width` object.

### Properties

#### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

#### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Value — Width of object

character vector

Width of object, such as an image or a table entry, specified in the form valueUnits, where Units is an abbreviation for the units. These abbreviations are valid:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points
- % — percent of table width

The resulting width of a column depends on the types of widths (absolute or fractional (%) of table width) that you use for each table entry. Setting the width of the entire column, or setting a table entry to resize to fit its contents also affect the resulting column width.

Table Entry Width Setting	Resulting Column Width
Two or more table entries set to different absolute widths	Maximum of the absolute widths
Two or more table entries set to different fractional widths, expressed as a percent (%) of the table width	Maximum of the fractional widths
Mixture of table entries set to absolute widths and fractional widths	For PDF output, maximum of the absolute widths. For all other output formats (for example, Word and HTML), the output application determines the maximum column width to use.
One or more table entry widths set to absolute widths or fractional widths, and the column width set using <code>mlreportgen.dom.TableColSpecGroup</code>	Maximum of the specified table entry widths and the column width
One or more table entry widths set to absolute widths or fractional widths, and <code>mlreportgen.dom.ResizeToFitContents</code> set to true	For PDF output, maximum of the table entry widths. For all other output formats, maximum of the resized-to-fit table entry widths.

## Examples

### Set Width and Other Formats for a Table

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);

t = Table(magic(5));
t.Style = {Border('inset','crimson','6pt'),...
           Width('50%')};

t.TableEntriesInnerMargin = '6pt';
t.TableEntriesHAlign = 'center';
t.TableEntriesVAlign = 'middle';
append(d,t);

close(d);
rptview('test',doctype);
```

### See Also

[mlreportgen.dom.Height](#) | [mlreportgen.dom.Table](#) | [mlreportgen.dom.Image](#) | [mlreportgen.dom.TableColSpecGroup](#)

### Topics

“Report Formatting Approaches” on page 13-17

# mlreportgen.finder.AxesFinder class

**Package:** mlreportgen.finder mlreportgen.finder mlreportgen.finder  
mlreportgen.finder

Find axes in MATLAB figure window

## Description

Use an object of the `mlreportgen.finder.AxesFinder` class to find all axes in a MATLAB figure window.

The `mlreportgen.finder.AxesFinder` class is a `handle` class.

## Class Attributes

`HandleCompatible` `true`

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`finder = mlreportgen.finder.AxesFinder(container)` creates an `mlreportgen.finder.AxesFinder` object and sets the `Container` property to a figure handle or path to a figure handle.

`finder = mlreportgen.finder.AxesFinder(Name, Value)` sets properties by using name-value arguments. Specify multiple name-value arguments in any order.

## Properties

### Container — Figure container

figure handle | string scalar | character vector

Figure container, specified as a:

- Character vector or string scalar that contains the path to a valid figure file
- Valid figure handle

### Properties — Properties of axes to find

{ } | cell array

Properties of the axes to find, specified as a cell array of name-value arguments. The finder returns only axes that have the specified properties with the specified values.

## Methods

### Public Methods

find	<code>results = find(AxesFinderObject)</code> finds axes in the figure specified by the finder. This method returns all axes as <code>mlreportgen.finder.AxesResult</code> objects. These result objects can be added directly to reports of type <code>mlreportgen.report.Report</code> or <code>slreportgen.report.Report</code> .
hasNext	<code>tf = hasNext(AxesFinderObject)</code> determines if the figure that the finder searches contains at least one of the specified axes to find. If the figure contains at least one axis, the <code>hasNext</code> method queues that axis as the next item that the <code>next</code> method will return. The <code>hasNext</code> method then returns true. Use the <code>next</code> method to obtain that axes. On subsequent calls, the <code>hasNext</code> method determines if the figure contains an axes that the <code>next</code> method has not yet retrieved. It queues the axes for the next method to retrieve and returns true. If there are no more axes to be retrieved, this method returns false. To search a container progressively for axes, use the <code>hasNext</code> method with the <code>next</code> method in a <code>while</code> loop.
next	<code>result = next(AxesFinderObject)</code> returns the next search result in the result queue created by the <code>hasNext</code> method as this method returns the axes that it finds, wrapped in an <code>mlreportgen.finder.AxesResult</code> object.

## Examples

### Find and Include Multiple Axes in a Report

This example shows how to use an `mlreportgen.finder.AxesFinder` object to find and include the axes in a given figure window.

Create a report.

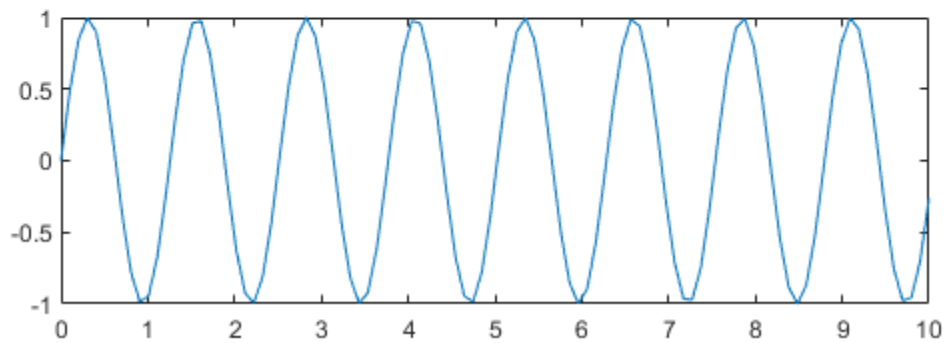
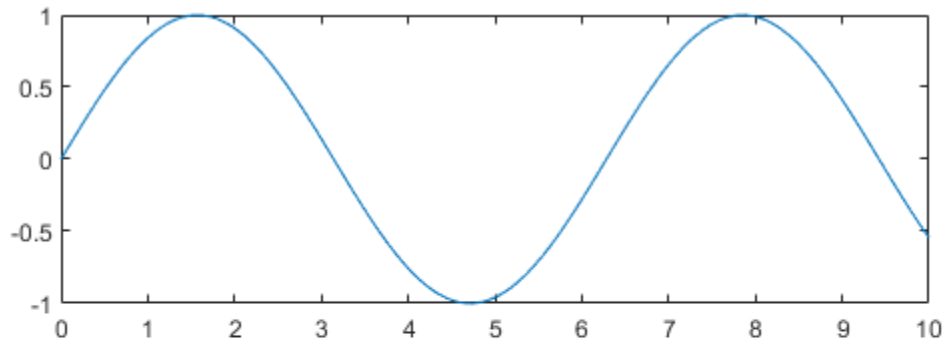
```
import mlreportgen.report.*
import mlreportgen.finder.*
rpt = Report("AxesFinder Example", "pdf");
```

Create a figure handle and plots to include in the report.

```
f = figure;
axes1 = subplot(2,1,1);
```



```
x = linspace(0,10);  
y1 = sin(x);  
plot(x,y1);  
  
axes2 = subplot(2,1,2);  
y2 = sin(5*x);  
plot(x,y2);
```

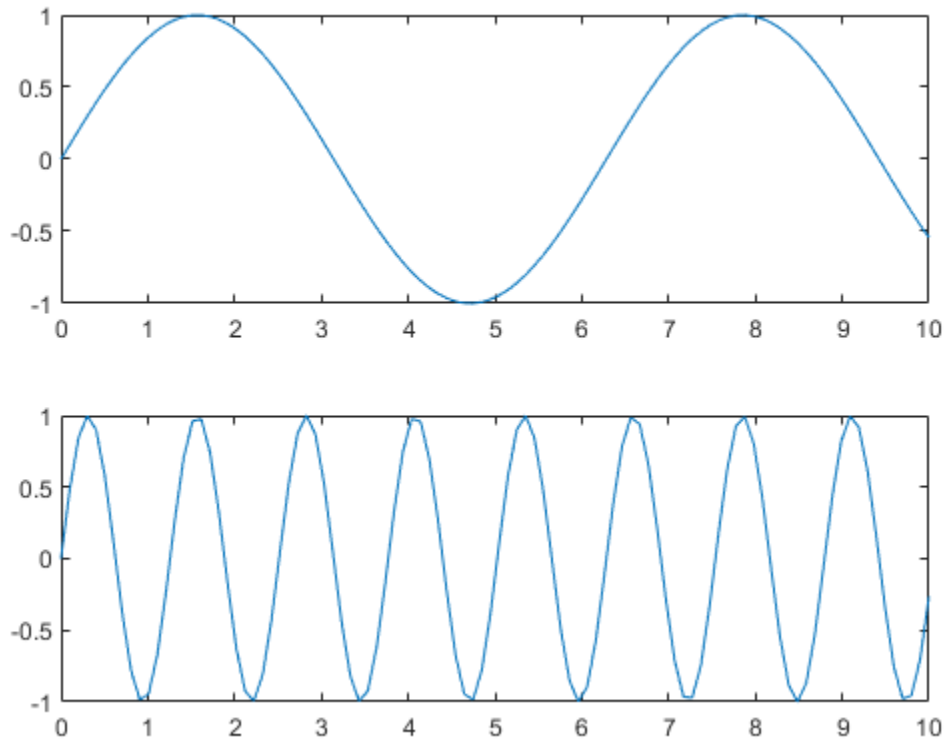


Use the `find` method from `mlreportgen.finder.AxesFinder` to search for all axes currently open in the given figure window.

```
customFinder = mlreportgen.finder.AxesFinder(f);  
results = customFinder.find;
```

Add the results to your report. Close and view the report.

```
append(rpt, results);
```



```
close(rpt);  
rptview(rpt);
```

### See Also

`mlreportgen.finder.AxesResult` | `mlreportgen.report.Axes` |  
`mlreportgen.report.Figure`

**Introduced in R2021b**

# mlreportgen.finder.AxesResult class

**Package:** mlreportgen.finder

**Superclasses:** mlreportgen.finder.Result

Result of figure axes find operation

## Description

Objects of mlreportgen.finder.AxesResult class store the result of searching for axes in a MATLAB figure.

The mlreportgen.finder.AxesResult class is a handle class.

### Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

You do not create an mlreportgen.finder.AxesResult object explicitly. When you use the findmethod of an mlreportgen.finder.AxesFinder object to search for an axes in a MATLAB figure, the method returns an AxesResult object for each axes that it finds.

## Properties

### Object — Axes handle

axes handle

Axes handle, returned as the handle to the axes.

#### Attributes:

GetAccess	public
SetAccess	protected

### Tag — Additional information

string scalar | character vector | object | ...

Additional information to add to this result. Set it to any type of value

#### Attributes:

GetAccess	public
SetAccess	public

## **Methods**

### **Public Methods**

`mlreportgen.finder.AxesResult.getReporter` Get axes reporter

### **See Also**

`mlreportgen.finder.AxesFinder` | `mlreportgen.report.Axes` |  
`mlreportgen.report.Figure`

### **Introduced in R2021b**

# mlreportgen.finder.Finder class

**Package:** mlreportgen.finder

Create A MATLAB Finder

## Description

mlreportgen.finder.Finder is an abstract class of finder objects to find result objects that can be added to an mlreportgen.report.Report object. Use this class as a basis for creating your own finder class.

## Properties

### Container — Container to be searched

depends on the container

Container to be searched by the finder. The data type depends on the type of container to be searched. For example, for a variable name, the data type is a character vector or string.

### Properties — Properties of objects to find

cell array

Properties of objects to find, specified as a cell array of name-value pairs. The finder returns only objects that have the specified properties with the specified values.

Example: `finder.Properties = {'Gain','5'}`

## Methods

`results = find(finder)` finds items in the container specified by the finder. This method returns the items it finds wrapped in result objects. These results objects can be added directly to reports of type mlreportgen.report.Report or slreportgen.report.Report. You can also add the results to a reporter that you then add to a report.

`tf = hasNext(finder)` determines if the container that the finder searches contains at least one of the specified items to find. If the container has at least one item, the hasNext method queues that item as the next item that the next method will return. The hasNext method then returns true. Use the next method to obtain that item. On subsequent calls, the hasNext method determines if the container has an item that the next method has not yet retrieved. It queues the item for the next method to retrieve and returns true. If there are no more items to be retrieved, this method returns false. To search a container progressively for items, use the hasNext method with the next method in a while loop.

`result = next(finder)` returns the next search result in the result queue that the hasNext method created. This method returns the item that it finds, wrapped in an mlreportgen.finder.Result object. To add tables of the item properties, add the result object to the report directly or add it to a reporter that you then add to a report. The reports to which you can add the results of this method must be of type mlreportgen.report.Report.

`tf = isIterating(finder)` checks whether the finder is iterating to find results. If `true`, you cannot change any of the finder properties.

`tf = mustNotBeIterating(finder)` or `tf = mustNotBeIterating(finder, propertyName)` validates that the finder is not iterating to find results. If `true`, the finder must not be iterating and you can change property values, etc. If `false`, the finder is iterating and you cannot change its properties, etc. The optional `propertyName` input is the property that is being modified and caused the error.

`tf = satisfyObjectPropertiesConstraint(finder, obj)` determines if the `obj` has a property that satisfies the `Properties` constraint specified by the finder.

`reset(finder)` resets the finder to its initial state, such that calling `next(finder)` returns the first result and resets the object states.

## Copy Semantics

Handle. To learn how handle classes affect copy operations, see [Copying Objects](#).

## See Also

`mlreportgen.finder.Result` | `mlreportgen.report.Report` |  
`mlreportgen.report.Reporter`

## Topics

“Create and Use a Custom Finder” on page 13-164

## Introduced in R2018a

# mlreportgen.finder.Result class

**Package:** mlreportgen.finder

Base class for finder results

## Description

mlreportgen.finder.Result is an abstract class that defines the properties and methods common to finder result classes. Use the `Result` class, or any descendent of the `Result` class, as the base for your custom result class.

The mlreportgen.finder.Result class is a handle class.

## Class Attributes

Abstract true

Hidden true

For information on class attributes, see “Class Attributes”.

## Properties

### Object — Object represented by result object

type depends on subclass implementation

Object represented by result object. The object type varies depending on the implementation of the subclass.

#### Attributes:

GetAccess	public
SetAccess	protected
Abstract	true

### Tag — Label

[] (default) | any MATLAB data type

Label, specified as any MATLAB data type.

#### Attributes:

GetAccess	public
SetAccess	public
Abstract	true

## Methods

### Public Methods

<code>getReporterLinkTargetID</code>	<code>linkTargetID = getReporterLinkTargetID(resObj)</code> returns a string that contains the default link target ID for the reporter object associated with the result object, <code>resObj</code> .
--------------------------------------	--

### Abstract Methods

Concrete subclasses must implement these abstract methods:

<code>getDefaultSummaryTableTitle</code>	<code>defTitle = getDefaultSummaryTableTitle(resObj)</code> returns the default title used by the <code>mlreportgen.report.SummaryTable</code> reporter with the result object, <code>resObj</code> .
<code>getDefaultSummaryProperties</code>	<code>defProps = getDefaultSummaryProperties(resObj)</code> returns a string array of the default list that contains properties reported on by the <code>mlreportgen.report.SummaryTable</code> reporter with the result object, <code>resObj</code> .
<code>getPropertyValues</code>	<code>propVals = getPropertyValues(resObj, propName)</code> returns a cell array that contains the values of the properties of the result object, <code>resObj</code> specified in the string array <code>propNames</code> .
<code>getReporter</code>	<code>reporterObj = getReporter(resObj)</code> returns the reporter object associated with the result object, <code>resObj</code> .

### See Also

`mlreportgen.finder.Finder` | `mlreportgen.report.Reporter` | `mlreportgen.report.Report`

**Introduced in R2018a**



# mlreportgen.finder.MATLABVariableFinder class

**Package:** mlreportgen.finder

**Superclasses:** mlreportgen.finder.Finder

MATLAB variable finder object

## Description

Use objects of the `mlreportgen.finder.MATLABVariableFinder` to find MATLAB variables by criteria. Then use the `find` or `next` methods to get the matching variables as `MATLABVariableResult` objects.

The `mlreportgen.finder.MATLABVariableFinder` class is a handle class.

## Class Attributes

`HandleCompatible` true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`varFinder = mlreportgen.finder.MATLABVariableFinder()` creates a `MATLABVariableFinder` object that finds variables in the base workspace.

`varFinder = mlreportgen.finder.MATLABVariableFinder(container)` creates a `MATLABVariableFinder` object that finds variables in the specified workspace, by setting the “Container” on page 12-0 property of the created `MATLABVariableFinder` object to the value of `container`.

`varFinder = mlreportgen.finder.MATLABVariableFinder(Name1=Value1, Name2=Value2)` creates a `MATLABVariableFinder` object and uses the name-value arguments to set the properties of the created `MATLABVariableFinder` object.

## Properties

### Container — Workspace in which to search for variables

"MATLAB" (default) | "Global" | string scalar | character vector

Workspace in which to search for variables, specified as one of these values:

Value	Workspace to Search
"MATLAB"	Base workspace
"Global"	Global workspace

Value	Workspace to Search
String scalar or character vector that contains the name or path of a MAT-file	The specified MAT-file

**Attributes**

GetAccess	public
SetAccess	public

Data Types: char | string

**Regexp — Enable regular expression name matching**

false or 0 (default) | true or 1

Whether to enable regular expression name matching, specified as a numeric or logical 1 (true) or 0 (false).

**Attributes**

GetAccess	public
SetAccess	public

Data Types: logical

**Name — Names of variables to search for**

string scalar | character vector

Names of variables to search for, specified as a string scalar or character vector that consists of the name of a specific variable or a regular expression. To use a regular expression to match the names of one or more MATLAB variables, set the Regexp property to true.

**Attributes**

GetAccess	public
SetAccess	public

Data Types: char | string

**IncludeReportVariables — Whether to include variables related to report generation**

false or 0 (default) | true or 1

Whether to include variables related to report generation, such as reporter or finder objects, specified as a logical 1 (true) or 0 (false).

**Attributes**

GetAccess	public
SetAccess	public

Data Types: logical

**Properties — Properties of variables to search for**

cell array

Properties of variables to search for, specified as a cell array of name-value arguments. Only variables that have the specified properties are searched for. The valid name-value arguments are:

Name	Value
"name"	Name of the variable, specified as a string scalar or a character array.
"class"	Data type of the variable, specified as a string scalar or a character array.
"size"	Dimensions of the variable, specified as a double array.
"bytes"	Number of bytes used for storing the variable in the computer memory, specified as a double scalar.
"sparse"	Whether the variable is a sparse matrix, specified as a logical.
"complex"	Whether the variable is a complex number, specified as a logical.
"global"	Whether the variable is global, specified as a logical.

### Attributes

```
GetAccess          public
SetAccess          public
```

Data Types: cell

### Methods

#### Public Methods

**Note** This finder provides two ways to get search results:

- 1 To return the search results as an array, use the `find` method. Add the results directly to a report or process the results in a `for` loop.
- 2 To iterate through the results one at a time, use the `hasNext` and `next` methods in a `while` loop.

Neither option has a performance advantage.

<code>find</code>	<code>resArr = find(varFinder)</code> finds variables that match the constraints specified by the <code>MATLABVariableFinder</code> object, and returns the variables as an array of <code>mlreportgen.finder.MATLABVariableResult</code> objects.
<code>hasNext</code>	<code>answer = hasNext(varFinder)</code> returns a logical 1 ( <code>true</code> ) if a variable is available to be retrieved by the <code>next</code> method. If no variable is available, <code>hasNext</code> returns a logical 0 ( <code>false</code> ).

next	theRes = next(varFinder) returns the next available variable as a MATLABVariableResult object. A variable is available if it matches varFinder and has not been previously retrieved by the next method. If no variable is available, then the next method returns an empty MATLABVariableResult object.
------	--

## Examples

### Find and Report on Specific Global Variables

This example shows how to create an `mlreportgen.finder.MATLABVariableFinder` object that finds global variables with names that begin with "my", then adds information about the matching variables to an `mlreportgen.report.Report` object.

Import these MATLAB Report API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
import mlreportgen.finder.*
import mlreportgen.dom.*
```

Define the global variables for this example.

```
global my_global_var1 my_global_var2 other_global_var;
my_global_var1 = "this is the content of my_global_var1";
my_global_var2 = "this is the content of my_global_var2";
other_global_var = "this is the content of other_global_var";
```

Create an `mlreportgen.report.Report` object of type PDF. Then create a title and append the title to the Report object.

```
theReport = Report("MATLABVariableFinder_globals_Example","pdf");
docHeader = Heading(1,"The global variables we found in this example are:");
append(theReport,docHeader);
```

Create an `mlreportgen.finder.MATLABVariableFinder` object that finds global variables with names that start with "my".

```
varFinder = MATLABVariableFinder(Container="Global",Name="^my",Regexp=true);
```

Use the `find` method to get an array of `mlreportgen.finder.MATLABVariableResult` objects that contain information about the matching variables.

```
finderResults = find(varFinder);
```

Append the matching variables to the report.

```
append(theReport,finderResults);
```

Close the Report object to create the PDF, then open the PDF in an editor.

```
close(theReport);
rptview(theReport);
```

Clear the global variables we used.

```
clear global my_global_var1 my_global_var2 other_global_var;
```

## Find and Report Individual Variables Iteratively

This example shows how to use the `next` and `hasNext` methods to retrieve MATLAB® variables from a MAT-file.

Import these MATLAB Report API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
import mlreportgen.finder.*
import mlreportgen.dom.*
```

Create an `mlreportgen.report.Report` object of type PDF. Then create a title and append it to the Report object.

```
theReport = Report("MATLABVariableFinder_MAT-file_Example","pdf");
docHeader = Heading(1,"The variables we found in the MAT-file for this example are:");
append(theReport,docHeader);
```

Create an `mlreportgen.finder.MATLABVariableFinder` object that finds variables in the MAT-file `Example_MAT_file.mat`.

```
varFinder = MATLABVariableFinder("Example_MAT_file.mat");
```

Use the `hasNext` and `next` methods to find the matching variables and append them to the Report object one at a time.

```
while(hasNext(varFinder))
    append(theReport,next(varFinder));
end
```

Close the Report object to create the PDF, then open the PDF in an editor.

```
close(theReport);
rptview(theReport);
```

## See Also

`mlreportgen.finder.MATLABVariableResult`

**Introduced in R2022a**

## mlreportgen.finder.MATLABVariableResult class

**Package:** mlreportgen.finder

**Superclasses:** mlreportgen.finder.Result

MATLAB variable search result object

### Description

An object of the `mlreportgen.finder.MATLABVariableResult` class is the result of a search for MATLAB variables. A `MATLABVariableResult` object contains information about a single variable. Each `MATLABVariableResult` object is associated with an `mlreportgen.report.MATLABVariable` reporter object that represents the same MATLAB variable. To report on the variable, add the `MATLABVariableResult` object to a generated report.

The `mlreportgen.finder.MATLABVariableResult` class is a handle class.

### Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

### Creation

You do not create `MATLABVariableResult` objects explicitly. The `MATLABVariableFinder` methods `find` and `next` create a `MATLABVariableResult` object for each variable they find.

### Properties

#### Object — Name of variable

string scalar

Name of the variable represented by this `MATLABVariableResult` object, returned as a string scalar.

#### Attributes

GetAccess public  
SetAccess protected

Data Types: string

#### Location — Workspace in which the variable is defined

"MATLAB" | "Global" | "MAT-File"

Workspace in which the variable is defined, specified as one of these values:

Value	Workspace
"MATLAB"	Base workspace

Value	Workspace
"Global"	Global workspace
"MAT-File"	A MAT-file

**Attributes**

GetAccess                                public  
SetAccess                                Restricts access

Data Types: string

**FileName — Name of MAT-File in which variable is defined**

string scalar | []

Name of the MAT-file in which the variable is defined, specified as a string scalar. If the variable is not defined in a MAT-file, FileName is empty.

**Attributes**

GetAccess                                public  
SetAccess                                Restricts access

Data Types: string

**Tag — Optional label**

[] (default) | any object

Optional label, specified as any type of MATLAB variable.

**Attributes**

GetAccess    public  
SetAccess    public

**Methods****Public Methods**

getReporter	reporter = getReporter(resObj) returns the mlreportgen.report.MATLABVariable reporter associated with the search result object, resObj.
getReporterLinkTargetID	linkID = getReporterLinkTargetID(resObj) returns the link target ID of the mlreportgen.report.MATLABVariable reporter object associated with the search result object, resObj. Use linkID to create a hyperlink from any part of the report to the MATLAB variable.

getDefaultSummaryTableTitle	defTitle = getDefaultSummaryTableTitle(resObj) returns the default summary table title for an object of the class MATLABVariableResult, resObj. This value is the default title for an mlreportgen.dom.SummaryTable object created from a MATLABVariableResult array. After you create a summary table, you can change the title.
getDefaultSummaryProperties	propList = getDefaultSummaryProperties(resObj) returns the list of default summary table properties for an object of the class MATLABVariableResult, resObj. The return value of this method, ["Name", "Size", "Bytes", "Class"], is the list of default properties for an mlreportgen.dom.SummaryTable object created from a MATLABVariableResult array. After you create a summary table, you can change the properties.

getPropertyValues Get property values of MATLAB variable search result object

## Examples

### Add Summary Tables of MATLAB® Variables to a Report

This example shows how to add summary tables with information about MATLAB® variables to a report.

Import these packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
import mlreportgen.finder.*
import mlreportgen.dom.*
```

Define variables and global variables of different data types and sizes in the base workspace.

```
global gvar_string gvar_str_arr_2x3 gvar_uint8_arr_9x3;
gvar_string = "this is my string scalar variable";
gvar_str_arr_2x3 = ["str 1.1", "str 1.2", "str 1.3"; "str 2.1", "str 2.2", "str 2.3"];
gvar_uint8_arr_9x3 = uint8(randi(255,9,3));

var_char_arr = 'this is my character vector variable';
var_double_3x5x6 = rand(3,5,6);
var_cell_arr_7x8x9 = num2cell(rand(7,8,9));
```

Create an mlreportgen.report.Report object of type PDF. Then create a title and append it to the Report object.



```
rpt = Report("Summary_of_variables_example","pdf");
docHeader = Heading(1,"Summary of variables we found:");
append(rpt,docHeader);
```

Use the `mlreportgen.finder.MATLABVariableFinder.find` method to find the variables in the base workspace, and create an array of `mlreportgen.finder.MATLABVariableResult` objects that represent the variables.

```
results = find(MATLABVariableFinder());
```

Create an `mlreportgen.report.SummaryTable` object with the result array. The default title of the table is the return value of the `getDefaultSummaryTableTitle` method, and the default list of properties to be reported in the table is the return value of the `getDefaultSummaryProperties` method. Leave the title of the table and list of properties to be reported in the table unchanged.

```
table1 = SummaryTable(results);
```

Create a second `SummaryTable` object with the result array. Then set the title of this table to "global or not", and set the properties to be reported by this table to "Name" and "Global".

```
table2 = SummaryTable(results);
table2.Title = "global or not";
table2.Properties = ["Name","Global"];
```

Append the two summary tables to the report. Then close and view the report.

```
append(rpt,table1);
append(rpt,table2);
close(rpt);
rptview(rpt);
```

### Link to MATLAB® Variable in Report

This example shows how to create a hyperlink to a variable in a report.

Import these packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
import mlreportgen.finder.*
import mlreportgen.dom.*
```

Define variables in the base workspace.

```
for index = 1:99
    evalin("base",sprintf("var_%0.3i='this is the content of var_%0.2i'",index,index));
end
```

Create an `mlreportgen.report.Report` object of type HTML. Then create a title and append it to the Report object.

```
rpt = Report("MATLABVariableResult_hyperlink_Example","html-file");
header = Heading(1,"These are the variables we found:");
append(rpt,header);
```

Use the `mlreportgen.finder.MATLABVariableFinder.find` method to find variables in the base workspace that have names beginning with "var", and create a vector of

`mlreportgen.finder.MATLABVariableResult` objects that represent the variables. Then append the result objects to the report.

```
results = find(MATLABVariableFinder(Name="^var"));
append(rpt,results);
```

Create an `mlreportgen.dom.InternalLink` object with the link target ID of result number 31. Then append the `InternalLink` object to the report.

```
targetID = getReporterLinkTargetID(results(31));
link = InternalLink(targetID,"Jump to result #31");
append(rpt,link);
```

Close and view the report.

```
close(rpt);
rptview(rpt);
```

Scroll to the bottom of the report to test the hyperlink.

## See Also

`mlreportgen.report.MATLABVariable` | `mlreportgen.finder.MATLABVariableFinder` | `mlreportgen.report.SummaryTable`

**Introduced in R2022a**

# mlreportgen.ppt.AutoFit class

**Package:** mlreportgen.ppt

Scale text to fit placeholder or text box

## Description

Use an mlreportgen.ppt.AutoFit format object to scale text to fit a placeholder or text box in a PPT API slide. Add an AutoFit object to the Style property of an mlreportgen.ppt.ContentPlaceholder, mlreportgen.ppt.TextBoxPlaceholder, or mlreportgen.ppt.TextBox object.

The mlreportgen.ppt.AutoFit class is a handle class.

## Creation

### Description

`autoFitObj = AutoFit()` scales text in a content placeholder or text box. Specify the scaling percentage by setting the FontScale property.

`autoFitObj = AutoFit(tf)` scales the text if `tf` is true. Specify the scaling percentage by setting the FontScale property.

`autoFitObj = AutoFit(tf, fontScale)` scales the text if `tf` is true. The text is scaled by the percentage specified by `fontScale`. See the FontScale property.

## Properties

### Value — Whether to scale text

true (default) | false

Whether to scale text to fit a placeholder or text box, specified as one of these values:

- true — Scales text (default)
- false — Does not scale text

### FontScale — Text font scaling value

'92.5%' (default) | string scalar | character vector

Text font scaling value, specified as a string scalar or character vector. This property specifies the percentage by which the text font size is scaled. The text is scaled one time using the specified percentage. If the text still does not fit the placeholder or text box, reduce the FontScale value until the text fits.

### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## Examples

### Scale Text to Fit Content Placeholder

To fit text in a content placeholder, create an `mlreportgen.ppt.AutoFit` format object with the `Value` property set to `true` and add the object to the `Style` property of an `mlreportgen.ppt.ContentPlaceholder` object. This example adds one slide where the text is not scaled and one slide where the text is scaled.

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myAutoFitPresentation.pptx");
open(ppt);
```

Add the first slide to the presentation. The content placeholder in this slide does not scale the text to fit.

```
slide1 = add(ppt,"Title and Content");
contentPlaceholder = find(slide1,"Content");
replace(contentPlaceholder,Paragraph(char(randi(25,1,1000)+97)));
contentPlaceholder.Style = [contentPlaceholder.Style {AutoFit(false)}];
```

Add the second slide to the presentation. The content placeholder in this slide scales the text to fit.

```
slide2 = add(ppt,"Title and Content");
contentPlaceholder = find(slide2,"Content");
replace(contentPlaceholder(1),Paragraph(char(randi(25,1,1000)+97)));
contentPlaceholder.Style = [contentPlaceholder.Style {AutoFit(true)}];
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

### See Also

`mlreportgen.ppt.ContentPlaceholder` | `mlreportgen.ppt.TextBoxPlaceholder` | `mlreportgen.ppt.TextBox`

### Topics

“Presentation Formatting Approaches” on page 14-18

“Presentation Format Inheritance” on page 14-21

**Introduced in R2020a**

## mlreportgen.ppt.BackgroundColor class

**Package:** mlreportgen.ppt

Background color of presentation element

### Description

Specifies the background color of these presentation element PPT API objects:

- TextBox
- TextBoxPlaceholder
- ContentPlaceholder
- TablePlaceholder
- Table
- TableRow
- TableEntry
- ColSpec
- TextBox

### Construction

`backgroundColorObj = BackgroundColor()` creates a white background.

`backgroundColorObj = BackgroundColor(color)` creates a background color object based on the specified CSS color name or hexadecimal RGB color value.

### Input Arguments

#### **color — Background color**

character vector

Background color, specified as a character vector. You can use:

- The name of a color, specified as a character vector. The name must be a CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- Hexadecimal RGB (red, green, blue) color value, specified as a character vector. Use the format `#RRGGBB`. Use `#` as the first character and two-digit hexadecimal numbers each for the red, green, and blue values. For example, `'#0000ff'` specifies blue.

### Output Arguments

#### **backgroundColorObj — Background color**

mlreportgen.ppt.BackgroundColor object

Background color for a report object, returned as an `mlreportgen.ppt.BackgroundColor` object.

## Properties

### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

### Value — CSS color name or hexadecimal RGB value for this color

character vector

The name of a color, specified as a character vector, using one of these values:

- A CSS color name. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- An RGB value, using a character vector having the format #RRGGBB. Use # as the first character and two-digit hexadecimal numbers each for the red, green, and blue values. For example, '#0000ff' specifies blue.

## Examples

### Use Background Colors in a Table

Create a table with different color rows and table entries.

Set up a presentation with a slide titled A Colorful Table.

```
import mlreportgen.ppt.*

ppt = Presentation('myBackground.pptx');
open(ppt);
slide1 = add(ppt, 'Title and Content');
replace(slide1, 'Title', 'A Colorful Table');
```

Define the table, specifying different colors for the top row and for the first entry in the second row.

```
table1 = Table();

row1 = TableRow();
row1.Style = {BackgroundColor('beige')};
row1entry1 = TableEntry();
p2 = Paragraph('Beige row');
append(row1entry1, p2);
row1entry2 = TableEntry();
p3 = Paragraph('More text');
```

```

append(row1entry2,p3);
append(row1,row1entry1);
append(row1,row1entry2);

row2 = TableRow();
row2entry1 = TableEntry();
row2entry1.Style = {BackgroundColor('yellow')};
p4 = Paragraph('yellow cell');
append(row2entry1,p4);
row2entry2 = TableEntry();
p5 = Paragraph('default white background');
append(row2entry2,p5);
append(row2,row2entry1);
append(row2,row2entry2);

append(table1,row1);
append(table1,row2);

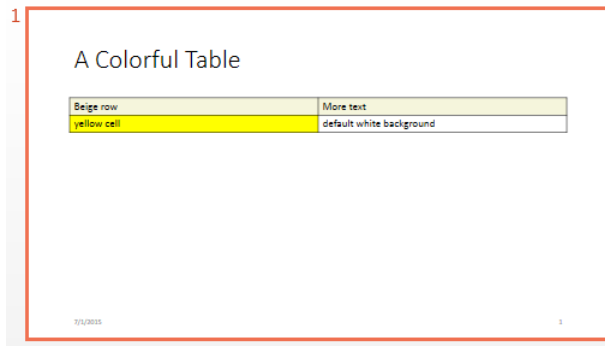
```

Replace the slide content with the table, generate the presentation, and open the myBackground presentation.

```

replace(slide1, 'Content', table1);
close(ppt);
rptview(ppt);

```



## See Also

mlreportgen.ppt.FontColor

## Topics

“Create and Format Tables” on page 14-69

“Presentation Formatting Approaches” on page 14-18



# mlreportgen.ppt.Bold class

**Package:** mlreportgen.ppt

Bold for text object

## Description

Specifies whether to use bold for a text object.

The mlreportgen.ppt.Bold class is a handle class.

## Creation

### Description

`boldObj = Bold()` creates a bold object that specifies to use bold for a text object.

`boldObj = Bold(value)` if `value` is `true`, creates a bold object that specifies to use bold for a text object. Otherwise, it creates a bold object that specifies to use regular weight text.

### Input Arguments

#### **value — Bold or regular weight for text**

`[]` (default) | logical value

Bold or regular weight for text, specified as a character vector. A setting of `false` (or `0`) uses regular weight text. A setting of `true` (or `1`) renders text in bold.

Data Types: `logical`

## Properties

### **Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### **Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

### **Value — Option to use bold or regular weight for a text object**

`[]` (default) | logical value

The possible values are:

- 0 — uses regular weight text
- 1 — renders text in bold

Data Types: `logical`

## Examples

### Create Paragraph With Bold and Regular-Weight Text

Create a presentation.

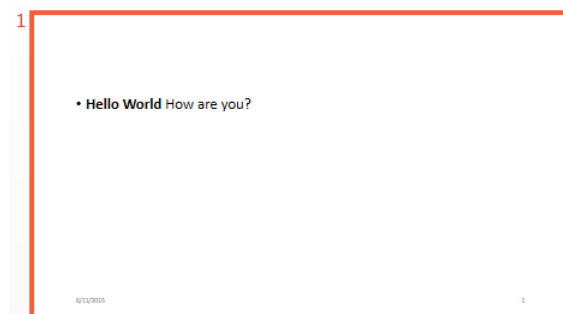
```
import mlreportgen.ppt.*  
  
ppt = Presentation('myBoldPresentation.pptx');  
open(ppt);  
  
titleSlide = add(ppt, 'Title and Content');
```

Create a paragraph and append text with bold text.

```
p = Paragraph('Hello World');  
p.Style = {Bold(true)};  
t = Text(' How are you?');  
t.Style = {Bold(false)};  
append(p,t);
```

Add the paragraph to the slide. Close and view the presentation.

```
replace(titleSlide, 'Content', p);  
  
close(ppt);  
rptview(ppt);
```



## See Also

`mlreportgen.ppt.Italic` | `mlreportgen.ppt.FontColor` | `mlreportgen.ppt.FontFamily` | `mlreportgen.ppt.FontSize`

## Topics

“Create and Format Text” on page 14-65

“Presentation Formatting Approaches” on page 14-18

**Introduced in R2015b**

## mlreportgen.ppt.Border class

**Package:** mlreportgen.ppt

Border of a table or table entry

### Description

Use an `mlreportgen.ppt.Border` format object to customize the border of a table or table entry in a PPT API presentation.

---

**Note** A conflict occurs if a border segment is shared by two table entries. For a conflicting horizontal border segment, Microsoft PowerPoint ignores the formats specified by the entry on the bottom. For a conflicting vertical border segment, PowerPoint ignores the formats specified by the entry on the right.

---

The `mlreportgen.ppt.Border` class is a handle class.

### Class Attributes

<code>ConstructOnLoad</code>	<code>true</code>
<code>HandleCompatible</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`mlreportgen.ppt.Border()` creates an empty `mlreportgen.ppt.Border` object. By default, PowerPoint renders table borders as thin, solid, black lines. Use the object properties to specify a different border width, style, or color.

`mlreportgen.ppt.Border(style)` sets the `Style`, `TopStyle`, `BottomStyle`, `LeftStyle`, and `RightStyle` properties to the value of `style`.

`mlreportgen.ppt.Border(style,color)` also sets the `Color`, `TopColor`, `BottomColor`, `LeftColor`, and `RightColor` properties to the value of `color`.

`mlreportgen.ppt.Border(style,color,width)` also sets the `Width`, `TopWidth`, `BottomWidth`, `LeftWidth`, and `RightWidth` properties to the value of `width`.

## Properties

### Style — Default style of border segments

`[]` (default) | `'none'` | `'solid'` | `'dot'` | ...

Default style of border segments, specified as a string scalar or character vector. Specify one of these values:

- 'none'
- 'solid'
- 'dot'
- 'dash'
- 'largeDash'
- 'dashDot'
- 'largeDashDot'
- 'largeDashDotDot'
- 'systemDash'
- 'systemDot'
- 'systemDashDot'
- 'systemDashDotDot'

### **Color — Default color of border segments**

[ ] (default) | character vector | string scalar

Default color of border segments, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

### **Width — Default width of border segments**

[ ] (default) | character vector | string scalar

Default width of border segments, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '3pt' specifies three points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

### **TopStyle — Style of top border segment**

[ ] (default) | 'none' | 'solid' | 'dot' | ...

Style of the top border segment, specified as a character vector or string scalar. Specify one of these values:

- 'none'
- 'solid'
- 'dot'
- 'dash'

- 'largeDash'
- 'dashDot'
- 'largeDashDot'
- 'largeDashDotDot'
- 'systemDash'
- 'systemDot'
- 'systemDashDot'
- 'systemDashDotDot'

**TopColor — Color of top border segment**

[] (default) | character vector | string scalar

Color of the top border segment, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

**TopWidth — Width of top border segment**

[] (default) | character vector | string scalar

Width of the top border segment, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '3pt' specifies three points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**BottomStyle — Style of bottom border segment**

[] (default) | 'none' | 'solid' | 'dot' | ...

Style of the bottom border segment, specified as a character vector or string scalar. Specify one of these values:

- 'none'
- 'solid'
- 'dot'
- 'dash'
- 'largeDash'
- 'dashDot'
- 'largeDashDot'
- 'largeDashDotDot'

- 'systemDash'
- 'systemDot'
- 'systemDashDot'
- 'systemDashDotDot'

### **BottomColor — Color of bottom border segment**

[ ] (default) | character vector | string scalar

Color of the bottom border segment, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

### **BottomWidth — Width of bottom border segment**

[ ] (default) | character vector | string scalar

Width of the bottom border segment, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '3pt' specifies three points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

### **LeftStyle — Style of left border segment**

[ ] (default) | 'none' | 'solid' | 'dot' | ...

Style of the left border segment, specified as a character vector or string scalar. Specify one of these values:

- 'none'
- 'solid'
- 'dot'
- 'dash'
- 'largeDash'
- 'dashDot'
- 'largeDashDot'
- 'largeDashDotDot'
- 'systemDash'
- 'systemDot'
- 'systemDashDot'
- 'systemDashDotDot'

**LeftColor — Color of left border segment**

[] (default) | character vector | string scalar

Color of the left border segment, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

**LeftWidth — Width of left border segment**

[] (default) | character vector | string scalar

Width of the left border segment, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '3pt' specifies three points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**RightStyle — Style of right border segment**

[] (default) | 'none' | 'solid' | 'dot' | ...

Style of the right border segment, specified as a character vector or string scalar. Specify one of these values:

- 'none'
- 'solid'
- 'dot'
- 'dash'
- 'largeDash'
- 'dashDot'
- 'largeDashDot'
- 'largeDashDotDot'
- 'systemDash'
- 'systemDot'
- 'systemDashDot'
- 'systemDashDotDot'

**RightColor — Color of right border segment**

character vector | string scalar

Color of the right border segment, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.



- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

### **RightWidth — Width of right border segment**

[ ] (default) | character vector | string scalar

Width of the right border segment, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '3pt' specifies three points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

### **Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### **Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## **Examples**

### **Customize the Border of a Table in a Presentation**

To customize the border or a border segment of a table or table entry in a PPT API presentation, specify the border style, color, and width in an mlreportgen.ppt.Border format object and add the object to the Style property of an mlreportgen.ppt.Table or mlreportgen.ppt.TabEntry object. This example specifies that the bottom border of the entry in the second row and second column is dashed, red, and has a width of three points.

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myTEBorders.pptx");
open(ppt);
```

Add a slide to the presentation.

```
slide = add(ppt, "Title and Table");
```

Create a table.

```
t = Table(magic(3));
```

Create a border object to customize the bottom border segment.

```
border = Border();
border.BottomStyle = "dash";
border.BottomColor = "red";
border.BottomWidth = "3pt";
```

Apply the custom border to the entry in the second column and second row.

```
tr2te2 = t.entry(2,2);
tr2te2.Style = [tr2te2.Style {border}];
```

Add a title and the table to the slide.

```
replace(slide, "Title", "Table entry with custom borders");
replace(slide, "Table", t);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here is the table in the generated presentation:

8	1	6
3	5	7
4	9	2

## See Also

`mlreportgen.ppt.Table` | `mlreportgen.ppt.TableEntry`

## Topics

“Create and Format Tables” on page 14-69

“Presentation Formatting Approaches” on page 14-18

“Presentation Format Inheritance” on page 14-21

**Introduced in R2020a**

## mlreportgen.ppt.ColSep class

**Package:** mlreportgen.ppt

Table column separators

### Description

Use an `mlreportgen.ppt.ColSep` format object to customize the lines that separate the columns of a table in a PPT API presentation.

The `mlreportgen.ppt.ColSep` class is a `handle` class.

### Class Attributes

<code>ConstructOnLoad</code>	<code>true</code>
<code>HandleCompatible</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`mlreportgen.ppt.ColSep()` creates an empty `mlreportgen.ppt.ColSep` object. By default, PowerPoint draws thin, solid, black lines between columns. Use the object properties to specify a different line width, style, or color.

`mlreportgen.ppt.ColSep(style)` sets the `Style` property to the value of `style`.

`mlreportgen.ppt.ColSep(style,color)` also sets the `Color` property to the value of `color`.

`mlreportgen.ppt.ColSep(style,color,width)` also sets the `Width` property to the value of `width`.

### Properties

#### Style — Column Separator style

`[]` (default) | `'none'` | `'solid'` | `'dot'` | ...

Column Separator style, specified as a character vector or string scalar. Specify one of these values:

- `'none'`
- `'solid'`
- `'dot'`
- `'dash'`
- `'largeDash'`
- `'dashDot'`

- 'largeDashDot'
- 'largeDashDotDot'
- 'systemDash'
- 'systemDot'
- 'systemDashDot'
- 'systemDashDotDot'

### **Color — Column separator color**

[ ] (default) | character vector | string scalar

Column separator color, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

### **Width — Column separator width**

[ ] (default) | character vector | string scalar

Column separator width, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '3pt' specifies three points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

### **Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### **Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## **Examples**

## Customize Table Column Separators

To customize the lines that separate table columns, set the properties of an `mlreportgen.ppt.ColSep` object and add the object to the `Style` property of an `mlreportgen.ppt.Table` object. This example makes the column separators dashed and red, with a line width of three points.

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myColSep.pptx");
open(ppt);
```

Add a slide to the presentation.

```
slide = add(ppt, "Title and Table");
```

Create a table and set custom column separators.

```
t = Table(magic(3));
t.Style = [t.Style {ColSep("dash", "red", "3pt")}];
```

Add a title and the table to the slide.

```
replace(slide, "Title", "Table with custom column separators");
replace(slide, "Table", t);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here is the table in the generated presentation:

Table with custom column separators		
8	1	6
3	5	7
4	9	2

12/4/2019 1

## See Also

`mlreportgen.ppt.Table` | `mlreportgen.ppt.RowSep` | `mlreportgen.ppt.Border`

**Topics**

“Create and Format Tables” on page 14-69

“Presentation Formatting Approaches” on page 14-18

“Presentation Format Inheritance” on page 14-21

**Introduced in R2020a**

## mlreportgen.ppt.ColSpec class

**Package:** mlreportgen.ppt

Formatting for table column

### Description

Use an object of the mlreportgen.ppt.ColSpec class to format the content of a table column in a PPT API presentation. Specify formats by setting the format properties, adding format objects to the Style property, or a combination of the two. See “Presentation Formatting Approaches” on page 14-18.

The mlreportgen.ppt.ColSpec class is a handle class.

### Class Attributes

ConstructOnLoad	true
HandleCompatible	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

colSpecObj = mlreportgen.ppt.ColSpec() creates an empty table column specification object.

colSpecObj = mlreportgen.ppt.ColSpec(colWidth) sets the Width property to colWidth.

## Properties

### Width — Column width

[] (default) | character vector | string scalar

Table column width, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '2in' specifies 2 inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

### Bold — Option to use bold for the column text

[] (default) | true or 1 | false or 0



Option to use bold for column text, specified as a numeric or logical 1 (`true`) or 0 (`false`). To make text bold, set this property to `true` or 1.

### **Font — Font family for column text**

[ ] (default) | character vector | string scalar

Font family for the column text, specified as a character vector or string scalar. Specify a font that appears in the font list in Microsoft PowerPoint. To see the font list, in PowerPoint, on the **Home** tab, in the **Font** group, click the arrow to the right of the font.

### **ComplexScriptFont — Font family for complex scripts**

[ ] (default) | character vector | string scalar

Font family for complex scripts, specified as a character vector or string scalar. Specify a font family to use when substituting in a locale that requires a complex script, such as Arabic or Asian, for rendering text.

### **FontColor — Font color of column text**

[ ] (default) | character vector | string scalar

Font color of the column text, specified as a character vector or string scalar, formatted as a CSS color name or a hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

### **FontSize — Font size of column text**

[ ] (default) | character vector | string scalar

Font size of the column text, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '11pt' specifies 11 points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

### **Italic — Option to use italic for column text**

[ ] (default) | true or 1 | false or 0

Option to use italic for column text, specified as a numeric or logical 1 (`true`) or 0 (`false`). If `Italic` is true or 1, text renders in italic. If `Italic` is false or 0, text renders as roman (straight).

### **Strike — Strikethrough style of column text**

[ ] (default) | 'single' | 'none' | 'double'

Strikethrough style of the column text, specified as a character vector or string scalar. Specify one of these values:

- 'single' — Single horizontal line
- 'none' — No strikethrough line
- 'double' — Double horizontal line

### **Subscript — Option to render column text as a subscript**

[ ] (default) | true or 1 | false or 0

Option to render column text as a subscript, specified as a numeric or logical 1 (true) or 0 (false). If Subscript is true or 1, text renders as a subscript. If Subscript is false or 0, text renders as regular text.

### **Superscript — Option to render column text as a superscript**

[ ] (default) | true or 1 | false or 0

Option to render column text as a superscript, specified as a numeric or logical 1 (true) or 0 (false). If Superscript is true or 1, text renders as a superscript. If Superscript is false or 0, text renders as regular text.

### **Underline — Underline style for column text**

[ ] (default) | 'single' | 'double' | ...

Underline style for column text, specified as a character vector or string scalar. Specify one of the values in the table.

Value	Description
'single'	Single underline
'double'	Double underline
'heavy'	Thick underline
'words'	Only words underlined (not spaces)
'dotted'	Dotted underline
'dottedheavy'	Thick, dotted underline
'dash'	Dashed underline
'dashheavy'	Thick, dashed underline
'dashlong'	Long, dashed underline
'dashlongheavy'	Thick, long, dashed underline
'dotdash'	Dot-dash underline
'dotdashheavy'	Thick, dot-dash underline
'dotdotdash'	Dot-dot-dash underline
'dotdotdashheavy'	Thick, dot-dot-dash underline
'wavy'	Wavy underline
'wavyheavy'	Thick, wavy underline
'wavydouble'	Wavy, double underline
'none'	No underline

### **BackgroundColor — Column background color**

[ ] (default) | character vector | string scalar

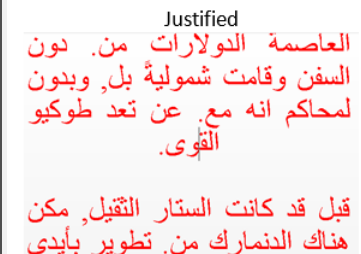
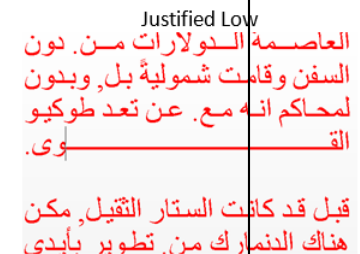
Background color, specified as a character vector or string scalar, formatted as a CSS color name or a hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

### HALign — Horizontal alignment of column content

[] (default) | 'center' | 'left' | ...

Horizontal alignment of the content in table entries of the column, specified as a character vector or string scalar. Specify one of the values in the table.

Value	Description
'center'	Centered
'left'	Left-justified
'right'	Right-justified
'justified'	Left and right-justified, spacing words evenly
'distributed'	Left and right-justified, spacing letters evenly
'thaiDistributed'	Left and right-justified Thai text, spacing characters evenly
'justifiedLow'	Justification for Arabic text <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;"> <p>Justified</p>  </div> <div style="text-align: center;"> <p>Justified Low</p>  </div> </div>

### VALign — Vertical alignment of column content

[] (default) | 'top' | 'bottom' | ...

Vertical alignment of the content in table entries of the column, specified as a character vector or string scalar. Specify one of the values in the table.

Value	Description
'top'	Vertically aligned to the top
'bottom'	Vertically aligned to the bottom of the table entry
'middle'	Vertically aligned to the middle of the table entry
'topCentered'	Vertically aligned to the top and horizontally aligned to the center of the table entry
'bottomCentered'	Vertically aligned to the bottom and horizontally aligned to the center of the table entry

Value	Description
'middleCentered'	Vertically aligned to the middle and horizontally aligned to the center of the table entry

### TextOrientation — Orientation of column text

[ ] (default) | 'horizontal' | 'down' | 'up'

Orientation of the column text, specified as a character vector or string scalar. Specify one of the values in the table.

Value	Description
'horizontal'	Text orientation is horizontal.
'down'	Text orientation is vertical, with the content rotated 90 degrees, clockwise.
'up'	Text orientation is vertical, with the content rotated 90 degrees, counterclockwise.

See “Specify Orientation of Column Text” on page 12-581.

### Style — Column formatting

cell array of PPT format objects

Column formatting, specified as a cell array of PPT API format objects. Formats that do not apply to a ColSpec object are ignored.

### Children — Children of this PPT API object

cell array of PPT objects

Child elements of this object, specified as a cell array of PPT objects. This property is read-only.

### Parent — Parent of this PPT API object

PPT object

Parent of this object, specified as a PPT object. This property is read-only.

### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

## Examples

### Specify Table Column Formatting

Create a presentation and add a slide.

```
import mlreportgen.ppt.*

ppt = Presentation('myColSpec.pptx');
open(ppt);
slide = add(ppt, 'Title and Content');
```

Create a table. Specify the width and background color of the first two columns of the table.

```
t = Table(magic(12));
t.Style = {HAlign('center')};

colSpecs(2) = ColSpec('2in');
colSpecs(1) = ColSpec('1in');
colSpecs(1).BackgroundColor = 'red';
colSpecs(2).BackgroundColor = 'green';
t.ColSpecs = colSpecs;
```

Add the table to the slide.

```
replace(slide, 'Content', t);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

144	9	3	141	140	6	7	137	136	10	11	133
13	131	130	16	17	127	126	20	21	123	122	24
25	119	118	28	29	115	114	32	33	111	110	36
108	38	39	105	104	42	43	101	100	46	47	97
96	50	51	93	92	54	55	89	88	58	59	85
61	83	82	64	65	79	78	68	69	75	74	72
73	71	70	76	77	67	66	80	81	63	62	84
60	86	87	57	56	90	91	53	52	94	95	49
48	98	99	45	44	102	103	41	40	106	107	37
109	35	34	112	113	31	30	116	117	27	26	120
121	23	22	124	125	19	18	128	129	15	14	132
12	134	135	9	8	138	139	5	4	142	143	1

### Specify Orientation of Column Text

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myTextOrientation.pptx");
open(ppt);
```

Add a slide to the presentation.

```
slide = add(ppt, "Title and Table");
```

Create a table.

```
table = Table({'Col 1', 'Col 2'; 'entry 1', 'entry 2'});
table.Height = "2in";
table.Style = [table.Style {VAlign("middleCentered")}];
```

Define the formatting for the table columns by using `mlreportgen.ppt.ColSpec` objects. Specify that the orientation of the text in the first column is vertical, rotated 90 degrees, counterclockwise. Specify that the orientation of the text in the second column is vertical, rotated 90 degrees, clockwise.

```
colSpecs(1) = ColSpec("1in");
colSpecs(1).TextOrientation = "up";
colSpecs(2) = ColSpec("1in");
colSpecs(2).TextOrientation = "down";
```

Assign the `ColSpec` objects to the `Table` object.

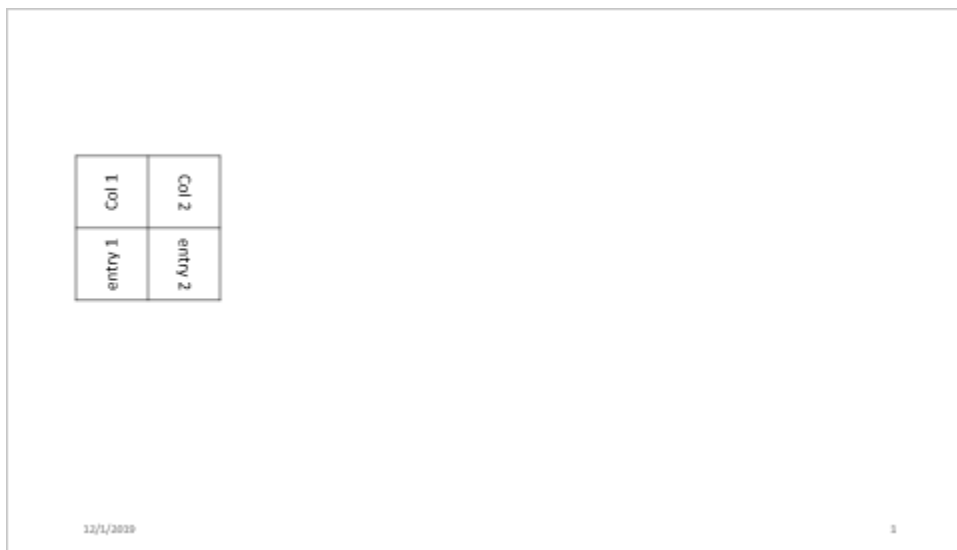
```
table.ColSpecs = colSpecs;
```

Add the table to the slide.

```
replace(slide, "Table", table);
```

Close and view the presentation

```
close(ppt);
rptview(ppt);
```



## See Also

`mlreportgen.ppt.ColWidth` | `mlreportgen.ppt.Bold` | `mlreportgen.ppt.FontFamily` | `mlreportgen.ppt.FontColor` | `mlreportgen.ppt.FontSize` | `mlreportgen.ppt.Italic` | `mlreportgen.ppt.Strike` | `mlreportgen.ppt.Subscript` | `mlreportgen.ppt.Superscript` | `mlreportgen.ppt.Underline` | `mlreportgen.ppt.BackgroundColor` | `mlreportgen.ppt.HAlign` | `mlreportgen.ppt.VAlign`

**Topics**

“Create and Format Tables” on page 14-69

“Presentation Formatting Approaches” on page 14-18

“Presentation Format Inheritance” on page 14-21

**Introduced in R2015b**

## mlreportgen.ppt.ColWidth class

**Package:** mlreportgen.ppt

Table column width

### Description

Width of a table column.

The mlreportgen.ppt.ColWidth class is a handle class.

## Creation

### Description

widthObj = ColWidth() creates a format object that specifies a column width of 0.25 inches.

widthObj = ColWidth(value) creates a column width object having the specified width.

### Input Arguments

#### value — Width of column

character vector

Width of column, specified in the form valueUnits, where Units is an abbreviation for the width units. These abbreviations are valid:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

## Properties

### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.



Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

### Value — Width of column

character vector

Width of column, specified in the form `valueUnits`, where `Units` is an abbreviation for the width units. These abbreviations are valid:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

## Examples

### Set Table Column Width

Create a presentation.

```
import mlreportgen.ppt.*

ppt = Presentation('myColWidth.pptx');
open(ppt);
slide1 = add(ppt, 'Title and Content');
```

Create a table and specify that the first column has a width of four inches.

```
C = {'wide column' 17 'aaaa' 4 5 6 7 8 9 10 11;...
    'long text string' 'bb' 1 3 5 7 9 11 13 15 17;...
    'more text' 1 2 3 4 5 6 7 8 9 10};

t = Table(C);
t.entry(1,1).Style = {ColWidth('4in')};
```

Replace the slide content with the table, generate the presentation, and open the `myColWidth` presentation.

```
replace(slide1, 'Content', t);
close(ppt);
rptview(ppt);
```

### See Also

`mlreportgen.ppt.ColSpec` | `mlreportgen.ppt.Table`

### Topics

“Create and Format Tables” on page 14-69

“Presentation Formatting Approaches” on page 14-18

**Introduced in R2015b**

## mlreportgen.ppt.ContentPlaceholder class

**Package:** mlreportgen.ppt

Placeholder for slide content

### Description

An object of the `mlreportgen.ppt.ContentPlaceholder` class represents a content placeholder in a slide. You can replace a content placeholder with a picture, table, or one or more paragraphs.

The PPT API creates a `ContentPlaceholder` object when you add a slide to a presentation and the slide layout has a content placeholder. In the default PPT API, these layouts have one or more content placeholders:

- 'Title and Content'
- 'Two Content'
- 'Comparison'
- 'Content with Caption'

To find a `ContentPlaceholder` object, use the `find` method of the slide that contains the content placeholder. To replace the content placeholder with a picture, table, or one or more paragraphs, use the `replace` method of the `ContentPlaceholder` object. For a picture or table, the `replace` method replaces the `ContentPlaceholder` object with an `mlreportgen.ppt.Picture` or `mlreportgen.ppt.Table` object, respectively. For paragraphs, the PPT API does not replace the `ContentPlaceholder` object. It adds `mlreportgen.ppt.Paragraph` objects as children of the `ContentPlaceholder` object.

When you replace a `ContentPlaceholder` object with a `Table` or `Picture` object, some of the `ContentPlaceholder` object properties do not apply to the replacement content. For details, see "Properties" on page 12-588.

---

**Note** If you replace a content placeholder with a picture, Microsoft PowerPoint adjusts the size of the content placeholder to accommodate the size of the picture. If you want the placeholder position and size to be fixed in a slide, use a slide with a picture placeholder. See `mlreportgen.ppt.PicturePlaceholder`.

---

The `mlreportgen.ppt.ContentPlaceholder` class is a `handle` class.

### Class Attributes

<code>HandleCompatible</code>	<code>true</code>
<code>ConstructOnLoad</code>	<code>true</code>

For information on class attributes, see "Class Attributes".

## Properties

### **Bold** — Whether to use bold for text

[ ] (default) | true | false

Whether to use bold for the text, specified as `true` or `false`. A setting of `true` renders the text in bold. A setting of `false` uses regular weight text.

This property applies only when you replace the content of the content placeholder with text.

Data Types: `logical`

### **Font** — Font family for text

[ ] (default) | character vector | string scalar

Font family for the text, specified as a character vector or string scalar. Specify a font that appears in the font list in Microsoft PowerPoint. To see the font list, in PowerPoint, on the **Home** tab, in the **Font** group, click the arrow to the right of the font.

This property applies only when you replace the content of the content placeholder with text.

### **ComplexScriptFont** — Font family for complex scripts

[ ] (default) | character vector | string scalar

Font family for complex scripts, specified as a character vector or string scalar. Specify a font family to use when substituting in a locale that requires a complex script, such as Arabic or Asian, for rendering text.

This property applies only when you replace the content of the content placeholder with text.

### **FontColor** — Font color for text

[ ] (default) | character vector | string scalar

Font color for the text, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use `#` as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, `'#0000ff'` specifies blue.

This property applies only when you replace the content of the content placeholder with text.

### **FontSize** — Font size of text

[ ] (default) | character vector | string scalar

Font size of the text, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, `'11pt'` specifies 11 points. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas

- pt — points

This property applies only when you replace the content of the content placeholder with text.

### **Italic — Whether to use italic for text**

[ ] (default) | true | false

Whether to use italic for the text, specified as true or false. A setting of true renders the text in italic.

This property applies only when you replace the content of the content placeholder with text.

Data Types: logical

### **Strike — Strikethrough style for text**

[ ] (default) | 'single' | 'none' | 'double'

Strikethrough style for the text, specified as one of these values:

- 'single' — Single horizontal line
- 'none' — No strikethrough line
- 'double' — Double horizontal line

This property applies only when you replace the content of the content placeholder with text.

### **Subscript — Whether to display text as a subscript**

[ ] (default) | true | false

Whether to display the text as a subscript, specified as true or false. A setting of true renders the text as a subscript.

This property applies only when you replace the content of the content placeholder with text.

Data Types: logical

### **Superscript — Whether to display text as a superscript**

[ ] (default) | true | false

Whether to display the text as a superscript, specified as true or false. A setting of true renders the text as a superscript.

This property applies only when you replace the content of the content placeholder with text.

Data Types: logical

### **Underline — Underline style for text**

[ ] (default) | 'single' | 'double' | ...

Underline style for the text, specified as one of these values:

Value	Description
'single'	Single underline
'double'	Double underline
'heavy'	Thick underline

Value	Description
'words'	Only words underlined (not spaces)
'dotted'	Dotted underline
'dottedheavy'	Thick, dotted underline
'dash'	Dashed underline
'dashheavy'	Thick, dashed underline
'dashlong'	Long, dashed underline
'dashlongheavy'	Thick, long, dashed underline
'dotdash'	Dot-dash underline
'dotdashheavy'	Thick, dot-dash underline
'dotdotdash'	Dot-dot-dash underline
'dotdotdashheavy'	Thick, dot-dot-dash underline
'wavy'	Wavy underline
'wavyheavy'	Thick, wavy underline
'wavydouble'	Wavy, double underline
'none'	No underline

This property applies only when you replace the content of the content placeholder with text.

### **BackgroundColor** – Background color for content placeholder

[ ] (default) | character vector | string scalar

Background color for the content placeholder, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

This property applies only when you replace the content of the content placeholder with text.

### **VAAlign** – Vertical alignment of text in the content placeholder

[ ] (default) | 'top' | 'bottom' | ...

Vertical alignment of the text in the content placeholder, specified as a character vector or string scalar. Specify one of the values in the table.

Value	Description
'top'	Vertically aligned to the top of the content placeholder
'bottom'	Vertically aligned to the bottom of the content placeholder
'middle'	Vertically aligned to the middle of the content placeholder

Value	Description
'topCentered'	Vertically aligned to the top and horizontally aligned to the center of the content placeholder
'bottomCentered'	Vertically aligned to the bottom and horizontally aligned to the center of the content placeholder
'middleCentered'	Vertically aligned to the middle and horizontally aligned to the center of the content placeholder

This property applies only when you replace the content of the content placeholder with text.

#### **Name — Content placeholder name**

character vector | string scalar

Content placeholder name, specified as a character vector or string scalar.

#### **X — Upper left x-coordinate of position of content placeholder**

character vector | string scalar

Upper left x-coordinate of the position of the content placeholder in the slide, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '5in' specifies 5 inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

#### **Y — Upper left y-coordinate of position of content placeholder**

character vector | string scalar

Upper left y-coordinate of the position of the content placeholder in the slide, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '5in' specifies 5 inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

#### **Width — Width of content placeholder**

character vector | string scalar

Width of the content placeholder, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '5in' specifies five inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

This property does not apply when the content placeholder is replaced with a picture.

**Height — Height of content placeholder**

character vector | string scalar

Height of the content placeholder, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '5in' specifies five inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

This property does not apply when the content placeholder is replaced with a picture.

**Style — Content placeholder formatting**

cell array of PPT format objects

Content placeholder formatting, specified as a cell array of PPT format objects.

Add format objects by concatenating the existing value of the `Style` property with a cell array that contains the format objects that you are adding. For example:

```
import mlreportgen.ppt.*
ppt = Presentation("My Presentation");
slide = add(ppt,"Title and Content");
content = find(slide,"Content");
placeholderObj = content(1);
placeholderObj.Style = [placeholderObj.Style {Bold(true),FontColor("red")}];
replace(placeholderObj,'Test');
```

See “Presentation Formatting Approaches” on page 14-18.

This property applies only when you replace the content of the content placeholder with text.

**Children — Children of this PPT API object**

cell array of PPT objects

Child elements of this object, specified as a cell array of PPT objects. This property is read-only.

**Parent — Parent of this PPT API object**

PPT object



Parent of this object, specified as a PPT object. This property is read-only.

### **Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### **Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS : ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## **Methods**

### **Public Methods**

add      Add paragraphs to content placeholder  
replace   Replace content placeholder or content

## **Examples**

### **Replace Content Placeholder with Content**

Add four `Title` and `Content` slides to a presentation. Replace the content placeholder of the first slide with text, the second slide with a table, the third slide with a picture, and the fourth slide with a multilevel list.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation.

```
ppt = Presentation("myContentPlaceholderPresentation.pptx");  
open(ppt);
```

Add a slide with a `Title` and `Content` layout.

```
slide1 = add(ppt,"Title and Content");
```

Use the `find` method of the slide object to find the placeholder object that has the name `Title`.

```
titlePlaceholderObj1 = find(slide1,"Title");
```

The `find` method returns an `mlreportgen.ppt.TextBoxPlaceholder` object.

Replace the placeholder content with the title text.

```
replace(titlePlaceholderObj1,"Content Replaced with Paragraph");
```

Use the `find` method of the slide object to find the placeholder object that has the name `Content`.

```
contentPlaceholderObj1 = find(slide1, "Content");
```

The `find` method returns an `mlreportgen.ppt.ContentPlaceholder` object.

Replace the placeholder content with a paragraph.

```
replace(contentPlaceholderObj1, Paragraph("This is my paragaph"));
```

Add a second slide with a `Title` and `Content` layout.

```
slide2 = add(ppt, "Title and Content");
```

Replace the placeholder for the title with the title text.

```
titlePlaceholderObj2 = find(slide2, "Title");  
replace(titlePlaceholderObj2, "Content Replaced with Table for Order 2 Magic Square");
```

Replace the content placeholder with a table.

```
contentPlaceholderObj2 = find(slide2, "Content");  
replace(contentPlaceholderObj2, Table(magic(2)));
```

Add a third slide with a `Title` and `Content` layout.

```
slide3 = add(ppt, "Title and Content");
```

Replace the placeholder for the title with the title text.

```
titlePlaceholderObj3 = find(slide3, "Title");  
replace(titlePlaceholderObj3, "Content Replaced with Picture of Peppers");
```

Replace the content placeholder with a picture.

```
contentPlaceholderObj3 = find(slide3, "Content");  
replace(contentPlaceholderObj3, Picture("peppers.png"));
```

Add a fourth slide with a `Title` and `Content` layout.

```
slide4 = add(ppt, "Title and Content");
```

Replace the placeholder for the title with the title text.

```
titlePlaceholderObj4 = find(slide4, "Title");  
replace(titlePlaceholderObj4, "Content Replaced with a Multilevel List");
```

Create content for a multilevel list. You can represent a multilevel list as a cell array that contains one or more cell arrays that represent sublists. Use an `mlreportgen.ppt.Paragraph` object to format an item in the list.

```
greenTea = Paragraph("Green Tea");  
greenTea.FontColor = "green";
```

```
multilevelContent = { ...  
    "Coffee", ...  
    "Tea", ...  
    { ...  
        "Black Tea", ...
```

```
        greenTea, ...  
    }, ...  
    "Milk", ...  
};
```

Replace the Content placeholder with the multilevel list content.

```
contentPlaceholderObj4 = find(slide4, "Content");  
replace(contentPlaceholderObj4, multilevelContent);
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

Here are the generated slides:

1

### Content Replaced with Paragraph

- This is my paragraph

11/16/2020

1

2

### Content Replaced with Table for Order 2 Magic Square

3	3
4	2

11/16/2020

2

3

### Content Replaced with Picture of Peppers



11/16/2020

3

4

### Content Replaced with a Multilevel List

- Coffee
- Tea
  - Black Tea
  - Green Tea
- Milk

11/16/2020

4

## Tips

- When you replace placeholder content in a presentation and then use the presentation as a template for a new presentation, the object that the PPT API creates for the content in the new presentation depends on the type of content, as shown in the table.

Replacement Content	Class of Object in New Presentation
picture	mlreportgen.ppt.TemplatePicture
table	mlreportgen.ppt.TemplateTable
paragraph	mlreportgen.ppt.ContentPlaceholder

For more information, see “Add and Replace Presentation Content” on page 14-58.

- To see the placeholder objects that the PPT API creates for a slide object, view the Children property of the slide. For example, when you add a Title and Content slide to a presentation, the Children property is an array that contains an mlreportgen.ppt.TextBoxPlaceholder object and an mlreportgen.ppt.ContentPlaceholder object.

```
ppt = mlreportgen.ppt.Presentation("test.pptx");
open(ppt);
slide = add(ppt,"Title and Content");
slide.Children(1)
```

ans =

TextBoxPlaceholder with properties:

```

        Bold: []
        Font: []
ComplexScriptFont: []
        FontColor: []
        FontSize: []
        Italic: []
        Strike: []
        Subscript: []
        Superscript: []
        Underline: []
        BackgroundColor: []
        VAlign: []
        Name: 'Title'
        X: []
        Y: []
        Width: []
        Height: []
        Style: []
        Children: []
        Parent: [1x1 mlreportgen.ppt.Slide]
        Tag: 'ppt.TextBoxPlaceholder:6:11'
        Id: '6:11'
```

```
slide.Children(2)
```

ans =

ContentPlaceholder with properties:

```
        Bold: []
        Font: []
ComplexScriptFont: []
        FontColor: []
        FontSize: []
        Italic: []
        Strike: []
        Subscript: []
        Superscript: []
        Underline: []
        BackgroundColor: []
        VAlign: []
        Name: 'Content'
        X: []
        Y: []
        Width: []
        Height: []
        Style: []
        Children: []
        Parent: [1x1 mlreportgen.ppt.Slide]
        Tag: 'ppt.ContentPlaceholder:7:12'
        Id: '7:12'
```

### **See Also**

`mlreportgen.ppt.Paragraph` | `mlreportgen.ppt.PicturePlaceholder` |  
`mlreportgen.ppt.TemplatePicture` | `mlreportgen.ppt.TablePlaceholder` |  
`mlreportgen.ppt.Table` | `mlreportgen.ppt.Picture` |  
`mlreportgen.ppt.TextBoxPlaceholder` | `mlreportgen.ppt.TemplateTable` |  
`getLayoutNames`

### **Topics**

“Access PowerPoint Template Elements” on page 14-32  
“Add and Replace Presentation Content” on page 14-58

### **Introduced in R2015b**

# mlreportgen.ppt.DebugMessage class

**Package:** mlreportgen.ppt

Debugging message

## Description

Creates debugging message text originating from the specified source object.

## Construction

`debugMsgObj = DebugMessage(text, sourceObject)` creates a debugging message with the specified text, originating from the specified source object.

### Input Arguments

**text — Message text**

character vector

The text to display for the message, specified as a character vector.

**sourceObject — PPT object from which message originates**

a PPT object

The PPT object from which the message originates, specified as a PPT object.

### Output Arguments

**debugMsgObj — Debugging message**

mlreportgen.ppt.DebugMessage object

Debug message, returned as an mlreportgen.ppt.DebugMessage object.

## Properties

**Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

**Source — Source object message originates from**

a PPT object

Source PPT object from which the message originates.

**Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

### Text — Text of the message

character vector

Message text, specified as a character vector.

## Methods

Use DebugMessage methods similar to how you use ProgressMessage methods.

Method	Purpose
formatAsHTML	Format message as HTML.
formatAsText	Format message as text.
passesFilter	Determine whether message passes filter.

## Examples

### Create a Debug Message

Create the presentation.

```
import mlreportgen.ppt.*;
pre = Presentation('myPresentation.pptx');
```

Create the listener and add it to the message dispatcher.

```
dispatcher = MessageDispatcher.getTheDispatcher;

dispatcher.Filter.ErrorMessagesPass = true;
dispatcher.Filter.ProgressMessagesPass = false;

l = addlistener(dispatcher, 'Message', ...
@(src, evtdata) disp(evtdata.Message.formatAsText));
```

Create the message and dispatch it before opening the presentation.

```
msg = ErrorMessage('Invalid slide', pre);
dispatch(dispatcher, msg);

open(pre);
```

Add content and close the presentation.

```
titleText = Text('This is a Title');
titleText.Style = {Bold};
replace(pre, 'Title', titleText);
```



```
close(pre);
```

Delete the listener to avoid duplicate reporting of message objects during a MATLAB session.

```
delete(l);
```

### **See Also**

`dispatch` | `mlreportgen.ppt.MessageEventData`

### **Topics**

“Display Presentation Generation Messages” on page 14-14

### **Introduced in R2015b**

## mlreportgen.ppt.ErrorMessage class

**Package:** mlreportgen.ppt

Error message

### Description

Specifies error message text originating from a specified source object.

### Construction

`errorMsgObj = ErrorMessage(text, sourceObject)` creates an error message with the specified text originating from the specified source object.

#### Input Arguments

**text — Message text**

character vector

The text to display for the message, specified as a character vector.

**sourceObject — The PPT object from which message originates**

a PPT object

The PPT object from which the message originates, specified as a PPT object

#### Output Arguments

**errorMsgObj — Error message**

mlreportgen.ppt.ErrorMessage object

Error message, returned as an mlreportgen.ppt.ErrorMessage object.

### Properties

**Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

**Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

**Source — Source object from which the message originates**

a PPT object

Source PPT object from which the message originates.

**Text — Text of message**

character vector

Message text, specified as a character vector.

**Methods**

Use ErrorMessage methods similar to how you use ProgressMessage methods.

Method	Purpose
formatAsHTML	Format message as HTML.
formatAsText	Format message as text.
passesFilter	Determine whether message passes filter.

**Examples****Create an Error Message**

Create the presentation.

```
import mlreportgen.ppt.*;
pre = Presentation('myPresentation.pptx');
```

Create the listener and add it to the message dispatcher.

```
dispatcher = MessageDispatcher.getTheDispatcher;

dispatcher.Filter.ErrorMessagesPass = true;
dispatcher.Filter.ProgressMessagesPass = false;

l = addlistener(dispatcher,'Message', ...
@(src, evtdata) disp(evtdata.Message.formatAsText));
```

Add an error to the program.

```
titleText = Text('This is a Title');
titleText.Style = {Bold};

replace(presentation,'Title',titleText);
```

Create the message and dispatch it. Then open the presentation.

```
msg = ErrorMessage('invalid slide',pre);
dispatch(dispatcher,msg);
```

```
open(pre);
```

Delete the listener to avoid duplicate reporting of message objects during a MATLAB session.

```
delete(l);
```

**See Also**

`dispatch | mlreportgen.ppt.MessageEventData`

**Topics**

“Display Presentation Generation Messages” on page 14-14

**Introduced in R2015b**

## mlreportgen.ppt.ExternalLink class

**Package:** mlreportgen.ppt mlreportgen.ppt

**Superclasses:** mlreportgen.ppt.Text

Hyperlink to location outside of presentation

### Description

Use an object of the mlreportgen.ppt.ExternalLink class to define a hyperlink to a location outside of the presentation.

The mlreportgen.ppt.ExternalLink class is a handle class.

### Class Attributes

HandleCompatible	true
ConstructOnLoad	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

externalLinkObj = ExternalLink() creates an empty mlreportgen.ppt.ExternalLink object.

externalLinkObj = ExternalLink(target,linkText) creates an mlreportgen.ppt.ExternalLink object with the Target property set to target and the Content property to linkText.

## Properties

### Target — URL of link target

'' (default) | character vector | string scalar

URL of the link target, specified as a character vector or string scalar. Specify the full URL. For example, include http://.

### Content — Link text

'' (default) | character vector | string scalar

Link text, specified as a character vector or string scalar.

### Bold — Whether to use bold for link text

[] (default) | true | false

Whether to use bold for the link text, specified as true or false. A setting of true renders text in bold.

**Font — Font family for link text**

[] (default) | character vector | string scalar

Font family for the link text, specified as a character vector or string scalar. Specify a font that appears in the font list in Microsoft PowerPoint. To see the font list, in PowerPoint, on the **Home** tab, in the **Font** group, click the arrow to the right of the font.

**ComplexScriptFont — Font family for complex scripts**

[] (default) | character vector | string scalar

Font family for complex scripts, specified as a character vector or string scalar. Specify a font family to use when substituting in a locale that requires a complex script, such as Arabic or Asian, to render text.

**FontColor — Font color for link text**

[] (default) | character vector | string scalar

Font color for the link text, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

**FontSize — Font size for link text**

[] (default) | character vector | string scalar

Font size for the link text, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '11pt' specifies 11 points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Italic — Whether to use italic for link text**

[] (default) | true | false

Whether to use italic for the link text, specified as true or false. A setting of true renders text in italic.

**Strike — Strikethrough style for link text**

[] (default) | 'single' | 'none' | 'double'

Strikethrough style for the link text, specified as one of these values:

- 'single' — Single horizontal line
- 'none' — No strikethrough line
- 'double' — Double horizontal line

**Subscript — Whether to display link text as a subscript**

[] (default) | true | false

Whether to display the link text as a subscript, specified as true or false. A setting of true renders text as a subscript.

**Superscript — Whether to display link text as a superscript**

[] (default) | true | false

Whether to display the link text as a superscript, specified as true or false. A setting of true renders text as a superscript.

**Underline — This property is ignored**

[] (default)

This property is ignored.

**Style — Link text formatting**

[] (default) | cell array of PPT format objects

Link text formatting, specified as a cell array of PPT format objects.

Add format objects by concatenating the existing value of the Style property with a cell array that contains the format objects that you are adding. For example:

```
link = mlreportgen.ppt.ExternalLink('link text');
link.Style = [link.Style {Bold(true)}];
```

See “Presentation Formatting Approaches” on page 14-18.

**Children — Children of this PPT API object (not used)**

[]

This property is not used.

**Parent — Parent of this PPT API object**

PPT object

Parent of this object, specified as a PPT object. This property is read-only.

**Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

**Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

## Methods

### Public Methods

clone	<p><code>objCopy = clone(obj)</code> returns a copy of the <code>mlreportgen.ppt.ExternalLink</code> object specified by <code>obj</code>.</p> <p>Use the <code>clone</code> method of an <code>ExternalLink</code> object the same way that you use the <code>clone</code> method of a <code>Paragraph</code> object.</p>
-------	--

## Examples

### Add an External Link to a Presentation

Create a presentation that includes a link to a web site.

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myExternalLinkPresentation.pptx");
open(ppt);
```

Add a slide to the presentation.

```
slide = add(ppt, "Title and Content");
```

Create an `mlreportgen.ppt.Paragraph` object.

```
p = Paragraph("This is a link to the ");
```

Create an `mlreportgen.ppt.ExternalLink` object and append it to the paragraph.

```
link = ExternalLink("https://www.mathworks.com", "MathWorks site");
append(p, link);
```

Replace the slide content with the paragraph.

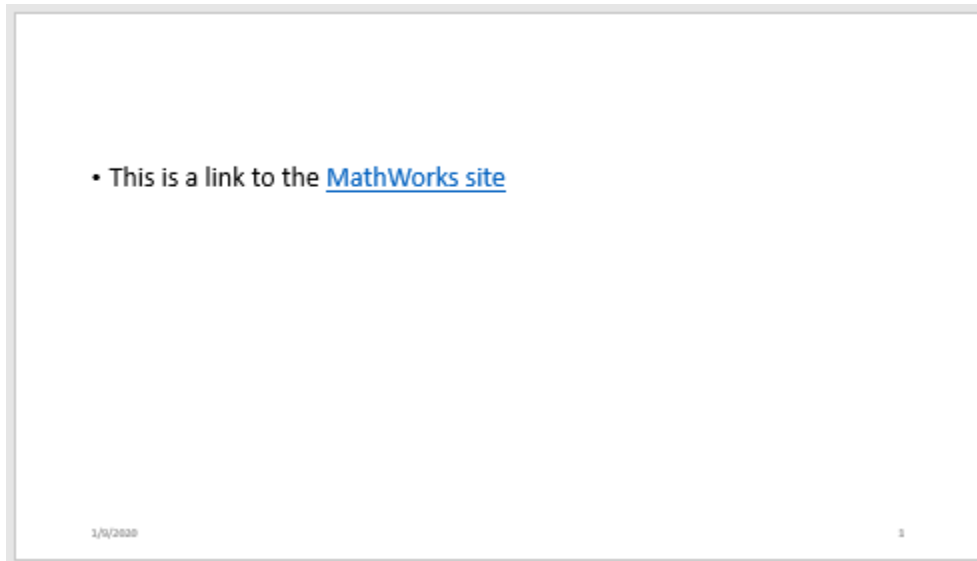
```
replace(slide, "Content", p);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here is the link in the generated slide.





## See Also

mlreportgen.ppt.InternalLink | mlreportgen.ppt.Text

## Topics

“Create and Format Links” on page 14-78

“Presentation Formatting Approaches” on page 14-18

“Presentation Format Inheritance” on page 14-21

## Introduced in R2015b

## mlreportgen.ppt.FontColor class

**Package:** mlreportgen.ppt

Font color

### Description

Format that specifies the font color in a presentation.

The mlreportgen.ppt.FontColor class is a handle class.

### Class Attributes

HandleCompatible	true
ConstructOnLoad	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

fontColorObj = mlreportgen.ppt.FontColor() creates a black font color object.

fontColorObj = mlreportgen.ppt.FontColor(value) sets the Value property to the color specified by value.

## Properties

### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

### Value — CSS color name or hexadecimal RGB value

character vector | string scalar

CSS color name or a hexadecimal RGB value, specified as a character vector or string scalar.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

## Examples

### Set the Text Color in a Presentation

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation('myFontColorPresentation.pptx');
open(ppt);
slide = add(ppt, 'Title and Content');
```

Create a paragraph and append text with colored text.

```
p = Paragraph('Hello World');

tRed = Text(' red text');
tRed.Style = {FontColor('red')};
append(p, tRed);

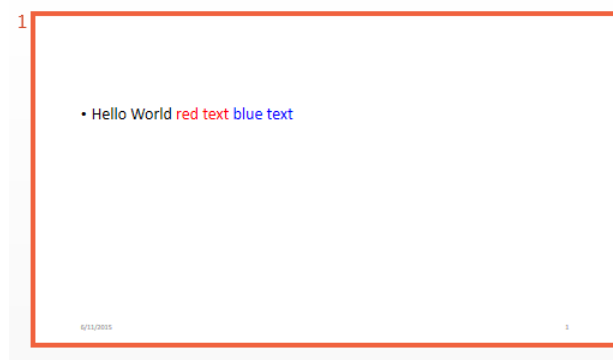
tBlue = Text(' blue text');
tBlue.Style = {FontColor('#0000ff')};
append(p, tBlue);
```

Add the paragraph to the slide.

```
replace(slide, 'Content', p);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```



### See Also

mlreportgen.ppt.FontFamily | mlreportgen.ppt.FontSize | mlreportgen.ppt.Bold | mlreportgen.ppt.Italic

**Topics**

“Create and Format Text” on page 14-65

“Presentation Formatting Approaches” on page 14-18

**Introduced in R2015b**

# mlreportgen.ppt.FontFamily class

**Package:** mlreportgen.ppt

Font family

## Description

Font family for presentation text.

## Construction

`fontFamilyObj = FontFamily()` specifies a Times New Roman font family.

`fontFamilyObj = FontFamily(fontStr)` specifies a font family.

`fontFamilyObj = FontFamily(fontStr, complexScriptFont)` specifies a font family to use for complex scripts.

## Input Arguments

### **fontStr — Font family**

character vector

Font family, specified as a character vector. Specify a font that appears in the PowerPoint list of fonts in the **Home** tab **Font** area.

### **complexScriptFont — Font family for complex scripts**

character vector

Font family for complex scripts, specified as a character vector. Specify a font family for substituting in a locale that requires a complex script (such as Arabic) for rendering text.

## Output Arguments

### **fontFamilyObj — Font family**

mlreportgen.ppt.FontFamily object

Font family, returned as an mlreportgen.ppt.FontFamily object.

## Properties

### **ComplexScriptFont — Font family for complex scripts**

character vector

Font family for complex scripts, specified as a character vector. Specify a font family for substituting in a locale that requires a complex script (such as Arabic) for rendering text.

### **Font — Font family**

character vector

Font family, specified as a character vector. Specify a font that appears in the PowerPoint list of fonts in **Home** tab **Font** area.

**Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

**Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## Examples

**Set the Font Family**

Create a presentation.

```
import mlreportgen.ppt.*

ppt = Presentation('myFontFamilyPresentation.pptx');
open(ppt);

titleSlide = add(ppt, 'Title and Content');

Create a paragraph and append text with text that uses the monospace font Courier New.

p = Paragraph('Use the ');

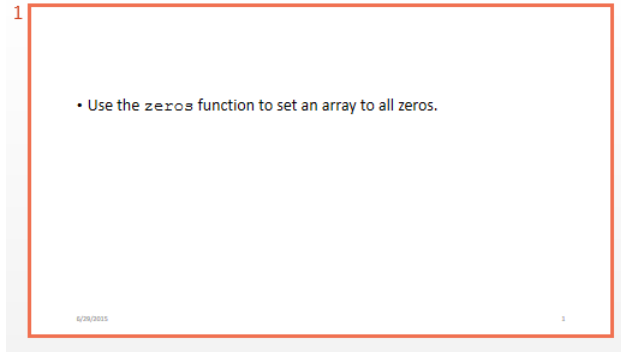
tFunc = Text('zeros');
tFunc.Style = {FontFamily('Courier New')};
append(p, tFunc);

tDesc = Text(' function to set an array to all zeros. ');
append(p, tDesc);
```

Add the paragraph to the slide. Close and view the presentation.

```
replace(titleSlide, 'Content', p);

close(ppt);
rptview(ppt);
```



## See Also

`mlreportgen.ppt.FontColor` | `mlreportgen.ppt.FontSize` | `mlreportgen.ppt.Text` | `mlreportgen.ppt.Paragraph`

## Topics

“Create and Format Text” on page 14-65  
“Create and Format Paragraphs” on page 14-67  
“Presentation Formatting Approaches” on page 14-18

**Introduced in R2015b**

## mlreportgen.ppt.FontSize class

**Package:** mlreportgen.ppt

Font size

### Description

Format that specifies the font size of text in a presentation.

The mlreportgen.ppt.FontSize class is a handle class.

### Class Attributes

HandleCompatible	true
ConstructOnLoad	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

fontSizeObj = mlreportgen.ppt.FontSize() creates a 12-point font.

fontSizeObj = mlreportgen.ppt.FontSize(value) sets the Value property to the font size specified by value.

## Properties

### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

### Value — Font size

'12pt' (default) | character vector | string scalar



Font size, specified as a character vector or string scalar, formatted as a numeric value followed by a unit of measurement. For example, '5in' specifies 5 inches. Use one of the following abbreviations for the unit of measurement:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

## Examples

### Set the Font Size of Text in a Presentation

Create a presentation.

```
import mlreportgen.ppt.*  
  
ppt = Presentation('myFontSizePresentation.pptx');  
open(ppt);  
slide = add(ppt, 'Title and Content');
```

Create a paragraph and append text in a large font size.

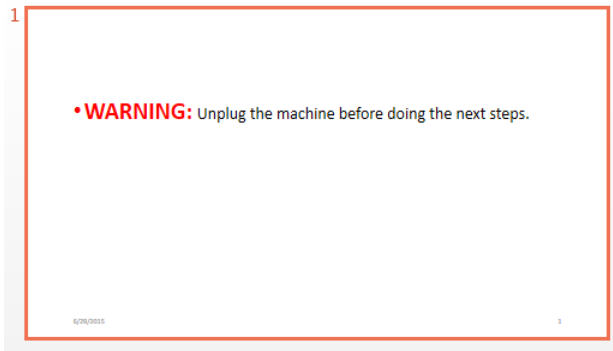
```
p = Paragraph();  
  
tWarning = Text('WARNING:');  
tWarning.Style = {FontSize('40pt'), Bold(true), FontColor('red')};  
append(p, tWarning);  
  
tDesc = Text(' Unplug the machine before doing the next steps. ');  
append(p, tDesc);
```

Add the paragraph to the slide.

```
replace(slide, 'Content', p);
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```



## See Also

`mlreportgen.ppt.FontFamily` | `mlreportgen.ppt.FontColor` | `mlreportgen.ppt.Text` | `mlreportgen.ppt.Paragraph`

## Topics

"Create and Format Text" on page 14-65

"Create and Format Paragraphs" on page 14-67

"Presentation Formatting Approaches" on page 14-18

## Introduced in R2015b

# mlreportgen.ppt.FlowDirection class

**Package:** mlreportgen.ppt

Table column flow direction

## Description

Use an mlreportgen.ppt.FlowDirection format object to specify the order of table columns in a PPT API presentation.

The mlreportgen.ppt.FlowDirection class is a handle class.

## Class Attributes

ConstructOnLoad	true
HandleCompatible	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

flowDirectionObj = mlreportgen.ppt.FlowDirection() sets the Value property to 'LeftToRight'.

flowDirectionObj = mlreportgen.ppt.FlowDirection(flow) sets the Value property to flow.

## Properties

### Value — Table column flow direction

'LeftToRight' (default) | 'RightToLeft'

Table column flow direction, specified as a character vector or string scalar. Specify one of these values:

- 'LeftToRight' — Orders columns from left to right
- 'RightToLeft' — Orders columns from right to left

### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## Examples

### Specify the Order of Table Columns

To specify the order of table columns, add an `mlreportgen.ppt.FlowDirection` format object to the `Style` property of an `mlreportgen.ppt.Table` object. This example specifies a right-to-left column order.

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myFlowDirection.pptx");
open(ppt);
```

Add a slide to the presentation.

```
slide = add(ppt, "Title and Content");
```

Create a table and specify the column flow direction.

```
t = Table({'entry(1,1)', 'entry(1,2)'; 'entry(2,1)', 'entry(2,2)'});
t.Style = [t.Style {FlowDirection("RightToLeft")}];
```

Add the table to the slide.

```
replace(slide, "Content", t);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

In the generated table, the second column appears first.

entry(1,2)	entry(1,1)
entry(2,2)	entry(2,1)

12/1/2019 1

## See Also

mlreportgen.ppt.Table

## Topics

“Create and Format Tables” on page 14-69

“Presentation Formatting Approaches” on page 14-18

“Presentation Format Inheritance” on page 14-21

## Introduced in R2020a

## mlreportgen.ppt.HAlign class

**Package:** mlreportgen.ppt

Horizontal alignment of paragraph

### Description

Specify the horizontal alignment of a paragraph.

### Construction

`alignObj = HAlign()` creates a horizontal alignment object having the value 'left'.

`alignObj = HAlign(value)` creates a horizontal alignment object having the specified value.

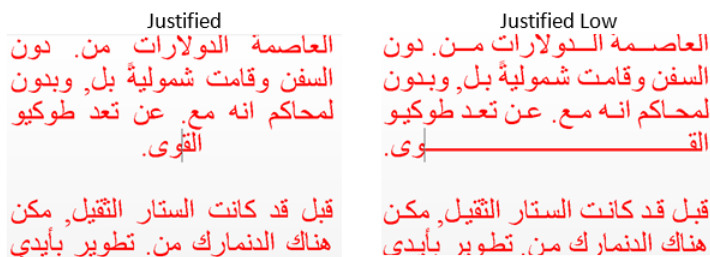
### Input Arguments

#### value — Horizontal alignment

character vector

Horizontal alignment, specified as one of these values:

- 'center' — Centered
- 'left' — Left justified
- 'right' — Right justified
- 'justified' — Left justified and right justified, spacing words evenly
- 'distributed' — Left justified and right justified, spacing letters evenly
- 'thaiDistributed' — Left justified and right justified Thai text, spacing characters evenly
- 'justifiedLow' — Justification for Arabic text



### Output Arguments

#### horizontalAlignObj — Horizontal alignment

mlreportgen.ppt.HAlign object

Horizontal alignment, returned as an mlreportgen.ppt.HAlign object.

## Properties

### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS : ID, where CLASS is the object class and ID is the value of the Id property of the object.

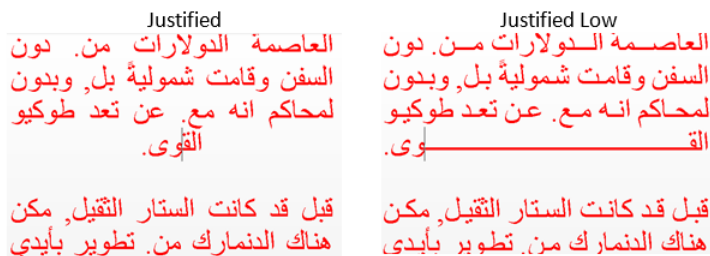
Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

### Value — Horizontal alignment

character vector

Horizontal alignment, specified as one of these values:

- 'center' — Centered
- 'left' — Left justified
- 'right' — Right justified
- 'justified' — Left justified and right justified, spacing words evenly
- 'distributed' — Left justified and right justified, spacing letters evenly
- 'thaiDistributed' — Left justified and right justified Thai text, spacing characters evenly
- 'justifiedLow' — Justification for Arabic text



## Examples

### Center a Title Paragraph

The presentation title page in the PPT API default template is to left-justify the title. This example overrides that default by centering the paragraph.

Create a presentation and add a title slide.

```
import mlreportgen.ppt.*
```

```
ppt = Presentation('myHAlignPresentation.pptx');  
open(ppt);  
titleSlide = add(ppt, 'Title Slide');
```

Create a centered paragraph.

```
p = Paragraph('Title for First Slide');  
p.Style = {HAlign('center')};
```

Add the paragraph to the slide, generate the presentation, and open myHAlignPresentation.

```
replace(titleSlide, 'Title', p);
```

```
close(ppt);  
rptview(ppt);
```

MATLAB:



## See Also

`mlreportgen.ppt.Paragraph` | `mlreportgen.ppt.VAlign`

## Topics

“Create and Format Paragraphs” on page 14-67

“Presentation Formatting Approaches” on page 14-18

**Introduced in R2015b**



## mlreportgen.ppt.InternalLink class

**Package:** mlreportgen.ppt mlreportgen.ppt

**Superclasses:** mlreportgen.ppt.Text

Hyperlink to a slide in a presentation

### Description

Use an object of the `mlreportgen.ppt.InternalLink` class to link from one slide to another slide in a presentation.

The `mlreportgen.ppt.InternalLink` class is a `handle` class.

### Class Attributes

<code>ConstructOnLoad</code>	<code>true</code>
<code>HandleCompatible</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`internalLinkObj = mlreportgen.ppt.InternalLink()` creates an empty `InternalLink` object.

`internalLinkObj = mlreportgen.ppt.InternalLink(targetSlideName,linkText)` creates an `InternalLink` object with the “Target” on page 12-0 property set to `targetSlideName` and the “Content” on page 12-0 property set to `linkText`.

`internalLinkObj = mlreportgen.ppt.InternalLink(targetSlideIndex,linkText)` creates an `InternalLink` object with the “Target” on page 12-0 property set to `targetSlideIndex` and the “Content” on page 12-0 property set to `linkText`.

## Properties

### Target — Target slide for hyperlink

`[]` (default) | integer | character vector | string scalar

Target slide of the hyperlink, specified as an integer, character vector, or string scalar. Use an integer to indicate the index of the target slide. Use a character vector or string scalar to indicate the name of the target slide.

### Content — Link text

`''` (default) | character vector | string scalar

Link text, specified as a character vector or string scalar.

**Bold — Whether to use bold for link text**

[] (default) | true | false

Whether to use bold for the link text, specified as `true` or `false`. A setting of `true` renders text in bold.

**Font — Font family for link text**

[] (default) | character vector | string scalar

Font family for the link text, specified as a character vector or string scalar. Specify a font that appears in the font list in Microsoft PowerPoint. To see the font list, in PowerPoint, on the **Home** tab, in the **Font** group, click the arrow to the right of the font.

**ComplexScriptFont — Font family for complex scripts**

[] (default) | character vector | string scalar

Font family for complex scripts, specified as a character vector or string scalar. Specify a font family to use when substituting in a locale that requires a complex script, such as Arabic or Asian, to render text.

**FontColor — Font color for link text**

[] (default) | character vector | string scalar

Font color for the link text, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use `#` as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, `'#0000ff'` specifies blue.

**FontSize — Font size for link text**

[] (default) | character vector | string scalar

Font size for the link text, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, `'11pt'` specifies 11 points. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**Italic — Whether to use italic for link text**

[] (default) | true | false

Whether to use italic for the link text, specified as `true` or `false`. A setting of `true` renders text in italic.

**Strike — Strikethrough style for link text**

[] (default) | 'single' | 'none' | 'double'

Strikethrough style for the link text, specified as one of these values:

- 'single' — Single horizontal line
- 'none' — No strikethrough line
- 'double' — Double horizontal line

### **Subscript — Whether to display link text as a subscript**

`[]` (default) | `true` | `false`

Whether to display the link text as a subscript, specified as `true` or `false`. A setting of `true` renders text as a subscript.

### **Superscript — Whether to display link text as a superscript**

`[]` (default) | `true` | `false`

Whether to display the link text as a superscript, specified as `true` or `false`. A setting of `true` renders text as a superscript.

### **Underline — This property is ignored**

`[]` (default)

This property is ignored.

### **Style — Link text formatting**

`[]` (default) | cell array of PPT format objects

Link text formatting, specified as a cell array of PPT format objects.

Add format objects by concatenating the existing value of the `Style` property with a cell array that contains the format objects that you are adding. For example:

```
link = mlreportgen.ppt.InternalLink(3, 'link text');
link.Style = [link.Style {Bold(true)}];
```

See “Presentation Formatting Approaches” on page 14-18.

### **Children — Children of this PPT API object (not used)**

`[]`

This property is not used.

### **Parent — Parent of this PPT API object**

PPT object

Parent of this object, specified as a PPT object. This property is read-only.

### **Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

**Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

**Methods****Public Methods**

clone	<p><code>objCopy = clone(obj)</code> returns a copy of the <code>mlreportgen.ppt.InternalLink</code> object specified by <code>obj</code>.</p> <p>Use the <code>clone</code> method of an <code>InternalLink</code> object the same way that you use the <code>clone</code> method of a <code>Paragraph</code> object.</p>
-------	--

**Examples****Link to Slide Using the Target Slide Name**

This example links to a slide using an `mlreportgen.ppt.InternalLink` object that specifies the target slide name.

Create the presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myPresentation1.pptx");
open(ppt);
```

Add a slide to the presentation.

```
slide1 = add(ppt, "Title and Content");
```

Choose a name to identify the target slide.

```
targetSlideName = "myTargetSlide";
```

Create a paragraph. Create an `InternalLink` object that specifies the target slide by name and append it to the paragraph.

```
p = Paragraph("This is a link to the slide with the name ");
linkObj = InternalLink(targetSlideName, targetSlideName);
append(p, linkObj);
```

Add the title and content to the slide.

```
replace(slide1, "Title", "First slide");
replace(slide1, "Content", p);
```

Add a second slide to the presentation.

```
slide2 = add(ppt, "Title and Content");
replace(slide2, "Title", "Second slide");
```

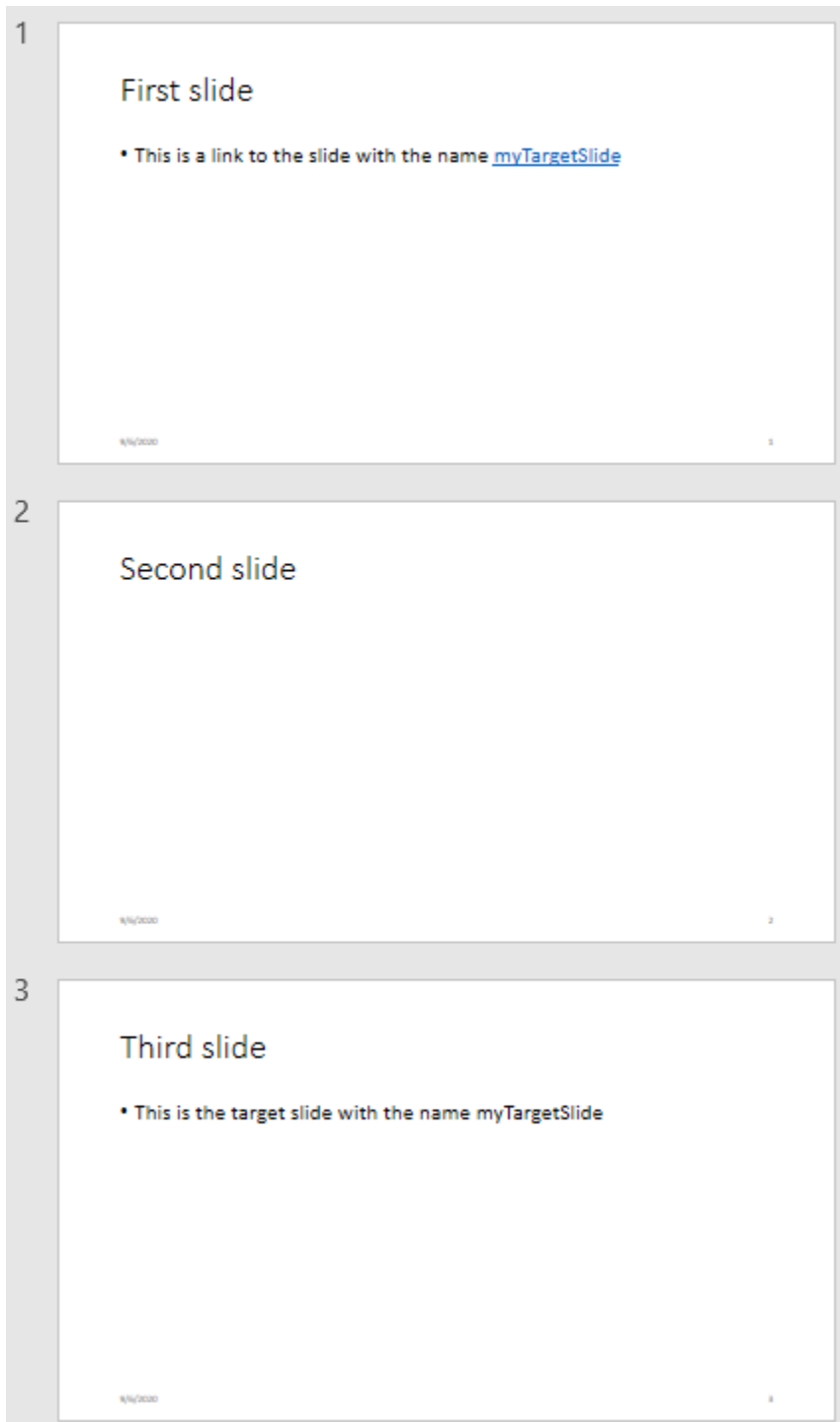
Add the target slide to the presentation. Set the Name property of the slide to the name specified in the InternalLink object.

```
slide3 = add(ppt,"Title and Content");  
slide3.Name = targetSlideName;  
replace(slide3,"Title","Third slide");  
content = strcat("This is the target slide with the name ",targetSlideName);  
replace(slide3,"Content",content);
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

Here are the generated slides:



### Link to a Slide Using the Target Slide Index

This example links to a slide using an `mReportgen.ppt.InternalLink` object that specifies the target slide number.

Create the presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myPresentation2.pptx");
open(ppt);
```

Add a slide to the presentation.

```
slide1 = add(ppt, "Title and Content");
```

Create a paragraph. Create an InternalLink object that specifies the target slide by its index and append the object to the paragraph.

```
p = Paragraph("This is a link to ");
link = InternalLink(3, "slide 3");
append(p, link);
```

Add the title and content to the slide.

```
replace(slide1, "Title", "First slide");
replace(slide1, "Content", p);
```

Add a slide 2 to the presentation.

```
slide2 = add(ppt, "Title and Content");
replace(slide2, "Title", "Second slide");
```

Add the target slide, slide 3, to the presentation.

```
slide3 = add(ppt, "Title and Content");
replace(slide3, "Title", "Third slide");
replace(slide3, "Content", "This is the target slide");
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here are the generated slides:

1

First slide

- This is a link to [slide 3](#)

8/25/2020 1

2

Second slide

8/25/2020 2

3

Third slide

- This is the target slide

8/25/2020 3



## **See Also**

mlreportgen.ppt.ExternalLink | mlreportgen.ppt.Paragraph | mlreportgen.ppt.Text

## **Topics**

“Create and Format Links” on page 14-78

“Presentation Formatting Approaches” on page 14-18

“Presentation Format Inheritance” on page 14-21

## **Introduced in R2021a**

## mlreportgen.ppt.Italic class

**Package:** mlreportgen.ppt

Italic for text object

### Description

Specifies whether to render text in italic.

### Construction

`italicObj = Italic()` creates a format object that specifies to render text in italic.

`italicObj = Italic(value)` creates a format object that specifies to render text in italic if `value` is `true`; otherwise, the text renders without italic.

### Input Arguments

**value — Option to use italic or text**

logical value

Option to use italic or text, specified as a logical. A setting of `true` (or `1`) renders text in italic.

Data Types: `logical`

### Output Arguments

**italicObj — Italic format object**

`mlreportgen.ppt.Italic` object

Italic format, returned as an `mlreportgen.ppt.Italic` object.

### Properties

**Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

**Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

**Value — Use italic or roman for text object**

[] (default) | logical value

The possible values are:

- 0 — uses roman (straight) text
- 1 — renders text in italic

Data Types: logical

**Examples****Create Paragraph with Italic and Regular Text**

Create a presentation.

```
import mlreportgen.ppt.*

ppt = Presentation('myItalicPresentation.pptx');
open(ppt);
titleSlide = add(ppt, 'Title and Content');
```

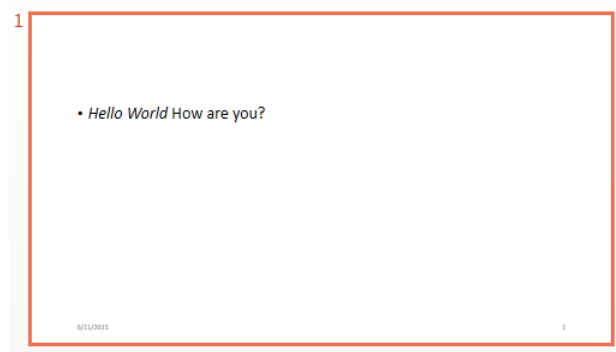
Create a paragraph and append text with italic and regular text.

```
p = Paragraph('Hello World');
p.Style = {Italic(true)};
t = Text(' How are you?');
t.Style = {Italic(false)};
append(p,t);
```

Add the paragraph to the slide. Close and view the presentation.

```
replace(titleSlide, 'Content', p);

close(ppt);
rptview(ppt);
```

**See Also**

mlreportgen.ppt.Bold

**Topics**

“Create and Format Text” on page 14-65

“Presentation Formatting Approaches” on page 14-18

**Introduced in R2015b**

# mlreportgen.ppt.MessageDispatcher class

**Package:** mlreportgen.ppt

PPT message dispatcher

## Description

Dispatcher for presentation generation status messages.

---

**Note** When you create a message dispatcher, the PPT API keeps the dispatcher until the end of the current MATLAB session. Delete message event listeners to avoid duplicate reporting of message objects during a MATLAB session.

---

## Properties

### Filter — Message filter

character vector

This read-only property specifies a filter that determines the types of messages the dispatcher dispatches. You can control the types of messages that are dispatched by setting the properties of the filter.

### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## Methods

Method	Purpose
dispatch	Dispatch a presentation generation status message.
mlreportgen.ppt.MessageDispatcher.getTheDispatcher	Get the message dispatcher.

## Examples

### Add and Dispatch a Progress Message

This example shows how to add a progress message to display when generating a presentation.

Create the presentation.

```
import mlreportgen.ppt.*;
pre = Presentation('myPresentation.pptx');
```

Create the listener and add it to the message dispatcher.

```
dispatcher = MessageDispatcher.getTheDispatcher;

dispatcher.Filter.ErrorMessagesPass = true;
dispatcher.Filter.ProgressMessagesPass = false;

l = addlistener(dispatcher, 'Message', ...
@(src, evtdata) disp(evtdata.Message.formatAsText));
```

Create the message and dispatch it before opening.

```
msg = ErrorMessage('Invalid slide',pre);
dispatch(dispatcher, msg);
```

```
open(pre);
```

Create an error in the program and dispatch the message before opening the presentation.

```
titleText = Text('This is a Title');
titleText.Style = {Bold};
replace(pre, 'Title', titleText);
```

```
close(pre);
```

Delete the listener to avoid duplicate reporting of message objects during a MATLAB session.

```
delete(l);
```

### See Also

[dispatch](#) | [mlreportgen.ppt.MessageDispatcher.getTheDispatcher](#) | [mlreportgen.ppt.MessageEventData](#) | [mlreportgen.ppt.MessageFilter](#)

### Topics

“Display Presentation Generation Messages” on page 14-14

### Introduced in R2015b

# mlreportgen.ppt.MessageEventData class

**Package:** mlreportgen.ppt

Holds message triggering message event

## Description

Contains the message that triggered a message event.

## Construction

`messageEventDataObj = MessageEventData(msg)` creates a message event data object that contains a PPT message, such as a message of type `mlreportgen.ppt.ProgressMessage`.

The PPT message dispatcher attaches an object of this type to a message event when it dispatches a message. Attaching the object enables message event listeners to retrieve the dispatched message. You need to create instances of this type only if you want to create your own message dispatcher.

## Input Arguments

### **msg — Message object**

message object

A message object, such as an `mlreportgen.ppt.ProgressMessage` object, that triggers a message event.

## Output Arguments

### **messageEventDataObj — Container for message that triggering message event**

`mlreportgen.ppt.MessageEventData` object

Container for message that triggers a message event, returned as an `mlreportgen.ppt.MessageEventData` object.

## Properties

### **Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### **Message — Message object**

message object

The value of this read-only property is a PPT message object, such as an `mlreportgen.ppt.ProgressMessage` object, that triggers a message event.

### **Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## Examples

### Capture Message Event Data

When you add a dispatcher, the PPT API creates the `evtdata` object, which is an `mlreportgen.ppt.MessageEventData` object.

Create the presentation.

```
import mlreportgen.ppt.*;
pre = Presentation('myPresentation.pptx');
```

Create the listener and add it to the message dispatcher.

```
dispatcher = MessageDispatcher.getTheDispatcher;

dispatcher.Filter.ErrorMessagesPass = true;
dispatcher.Filter.ProgressMessagesPass = false;

l = addlistener(dispatcher, 'Message', ...
@(src, evtdata) disp(evtdata.Message.formatAsText));
```

Create the message and dispatch it. Then open the presentation.

```
msg = ErrorMessage('Invalid slide',pre);
dispatch(dispatcher, msg);
```

```
open(pre);
```

Create an error in the program and dispatch the message before opening.

```
titleText = Text('This is a Title');
titleText.Style = {Bold};
replace(pre, 'Title', titleText);
```

```
close(pre);
```

### See Also

`dispatch`

### Topics

“Display Presentation Generation Messages” on page 14-14



# mlreportgen.ppt.MessageFilter class

**Package:** mlreportgen.ppt

Filter to control message dispatcher

## Description

Filter for messages dispatched by the message dispatcher.

## Properties

### **DebugMessagePass — Pass or block debug messages**

logical value

Pass or block debug messages, specified as a logical.

- `true` — Pass debug messages.
- `false` — Block debug messages.

Data Types: `logical`

### **ErrorMessagePass — Pass or block error messages**

logical value

- `true`— Pass error messages.
- `false`— Block error messages.

Data Types: `logical`

### **GlobalFilter — Pass or block all messages**

logical value

- `true`— Pass all messages.
- `false`— Block all messages.

Data Types: `logical`

### **ProgressMessagePass — Pass or block progress messages**

logical value

- `true`— Pass progress messages.
- `false`— Block progress messages.

Data Types: `logical`

### **GlobalFilter — Pass or block all messages**

logical value

- `true`— Pass all messages.
- `false`— Block all messages.

Data Types: logical

### **SourceFilter — Pass messages only for this PPT object**

PPT object

Pass messages only for this PPT object, specified as a PPT object. Pass messages only from the specified PPT object if the messages meet the other filter conditions specified by this `MessageFilter` object.

## **Examples**

### **Filter Messages**

Create the presentation.

```
import mlreportgen.ppt.*;
pre = Presentation('myPresentation.pptx');
```

Create the listener and add it to the message dispatcher.

```
dispatcher = MessageDispatcher.getTheDispatcher;

dispatcher.Filter.ErrorMessagePass = true;
dispatcher.Filter.ProgressMessagesPass = false;

l = addlistener(dispatcher, 'Message', ...
@(src, evtdata) disp(evtdata.Message.formatAsText));
```

Create the message and dispatch it before opening.

```
msg = ErrorMessage('Invalid slide',pre);
dispatch(dispatcher, msg);

open(pre);
```

Create an error in the program and dispatch the message. Then open the presentation.

```
titleText = Text('This is a Title');
titleText.Style = {Bold};
replace(pre, 'Title',titleText);

close(pre);
```

### **See Also**

`dispatch` | `mlreportgen.ppt.MessageEventData`

### **Topics**

“Display Presentation Generation Messages” on page 14-14

**Introduced in R2015b**

# mlreportgen.ppt.Paragraph class

**Package:** mlreportgen.ppt

Formatted block of text (paragraph)

## Description

Use an object of the mlreportgen.ppt.Paragraph class to include a paragraph in a PPT API presentation. A paragraph can contain text or mlreportgen.ppt.Text, mlreportgen.ppt.InternalLink, or mlreportgen.ppt.ExternalLink objects.

The mlreportgen.ppt.Paragraph class is a handle class.

### Class Attributes

HandleCompatible	true
ConstructOnLoad	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

paragraphObj = mlreportgen.ppt.Paragraph() creates an empty paragraph object.

paragraphObj = mlreportgen.ppt.Paragraph(text) creates a paragraph that contains an mlreportgen.ppt.Text object with the text specified by text.

paragraphObj = mlreportgen.ppt.Paragraph(pptElementObj) creates a paragraph that contains the text, internal link, or external link specified by pptElementObj.

### Input Arguments

#### text — Paragraph text

character vector | string scalar

Paragraph text, specified as a character vector or string scalar.

#### pptElementObj — Presentation element to include in paragraph

mlreportgen.ppt.Text object | mlreportgen.ppt.InternalLink | mlreportgen.ppt.ExternalLink object

Presentation element to include in the paragraph, specified as an mlreportgen.ppt.Text, mlreportgen.ppt.InternalLink, or mlreportgen.ppt.ExternalLink object.

## Properties

#### Bold — Whether to use bold for text

[] (default) | true | false

Whether to use bold for the text in this paragraph, specified as `true` or `false`. A setting of `false` uses regular weight text. A setting of `true` renders text in bold.

**Font — Font family for text**

[] (default) | character vector | string scalar

Font family for the text in this paragraph, specified as a character vector or string scalar. Specify a font that appears in the font list in Microsoft PowerPoint. To see the font list, in PowerPoint, on the **Home** tab, in the **Font** group, click the arrow to the right of the font.

**ComplexScriptFont — Font family for complex scripts**

[] (default) | character vector | string scalar

Font family for complex scripts, specified as a character vector or string scalar. Specify a font family to use when substituting in a locale that requires a complex script, such as Arabic or Asian, for rendering text.

**FontColor — Font color for text**

[] (default) | character vector | string scalar

Font color for the text in this paragraph, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use `#` as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, `'#0000ff'` specifies blue.

**FontSize — Font size of text**

[] (default) | character vector | string scalar

Font size of the text in this paragraph, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, `'11pt'` specifies 11 points. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**Italic — Whether to use italic for text**

[] (default) | `true` | `false`

Whether to use italic for the text in this paragraph, specified as `true` or `false`. A setting of `true` renders text in italic.

**Strike — Strikethrough style for text**

[] (default) | `'single'` | `'none'` | `'double'`

Strikethrough style for the text in this paragraph, specified as one of these values:

- 'single' — Single horizontal line
- 'none' — No strikethrough line
- 'double' — Double horizontal line

### **Subscript — Whether to display text as a subscript**

[ ] (default) | true | false

Whether to display the text in this paragraph as a subscript, specified as `true` or `false`. A setting of `true` renders text as a subscript.

### **Superscript — Whether to display text as a superscript**

[ ] (default) | true | false

Whether to display the text in this paragraph as a superscript, specified as `true` or `false`. A setting of `true` renders text as a superscript.

### **Underline — Underline style for text**

[ ] (default) | 'single' | 'double' | ...

Underline style for the text in this paragraph, specified as one of these values:

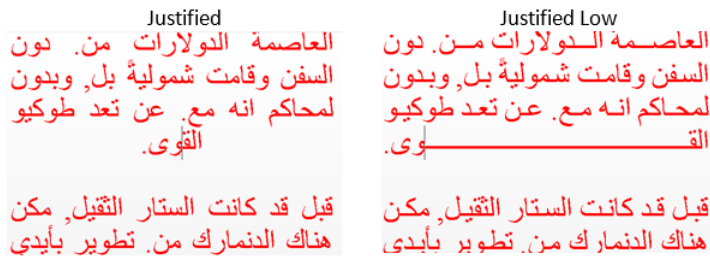
Value	Description
'single'	Single underline
'double'	Double underline
'heavy'	Thick underline
'words'	Only words underlined (not spaces)
'dotted'	Dotted underline
'dottedheavy'	Thick, dotted underline
'dash'	Dashed underline
'dashheavy'	Thick, dashed underline
'dashlong'	Long, dashed underline
'dashlongheavy'	Thick, long, dashed underline
'dotdash'	Dot-dash underline
'dotdashheavy'	Thick, dot-dash underline
'dotdotdash'	Dot-dot-dash underline
'dotdotdashheavy'	Thick, dot-dot-dash underline
'wavy'	Wavy underline
'wavyheavy'	Thick, wavy underline
'wavydouble'	Wavy, double underline
'none'	No underline

### **HALign — Horizontal alignment of text**

[ ] (default) | character vector | string scalar

Horizontal alignment of the text in this paragraph, specified as one of these values:

- 'center' — Centered
- 'left' — Left justified
- 'right' — Right justified
- 'justified' — Left justified and right justified, spacing words evenly
- 'distributed' — Left justified and right justified, spacing letters evenly
- 'thaiDistributed' — Left justified and right justified Thai text, spacing characters evenly
- 'justifiedLow' — Justification for Arabic text



### Level — Indentation level of paragraph

[ ] (default) | integer in the range [1,9]

Indentation level of this paragraph, specified as an integer in the range [1,9]. The value 1 indicates a top-level paragraph with no indentation.

### Style — Paragraph formatting

cell array of PPT format objects

Paragraph formatting, specified as a cell array of PPT format objects.

Add format objects by concatenating the existing value of the `Style` property with a cell array that contains the format objects that you are adding. For example:

```
para = mlreportgen.ppt.Paragraph('my paragraph');
para.Style = [para.Style {Bold(true),FontColor('red')}]
```

See “Presentation Formatting Approaches” on page 14-18.

### Children — Children of this PPT API object

cell array of PPT objects

Child elements of this object, specified as a cell array of PPT objects. This property is read-only.

### Parent — Parent of this PPT API object

PPT object

Parent of this object, specified as a PPT object. This property is read-only.

### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

### **Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

## **Methods**

### **Public Methods**

append Append content to paragraph  
clone Copy PPT API paragraph

## **Examples**

### **Add Paragraphs to Presentation Slides**

Create a presentation with two slides.

```
import mlreportgen.ppt.*;

ppt = Presentation('myParagraphPresentation.pptx');
open(ppt);

add(ppt, 'Title Slide');
add(ppt, 'Title and Content');
```

Create an `mlreportgen.ppt.Paragraph` object to use for the title. Make the text bold and red.

```
p = Paragraph('My Title');
p.Bold = true;
p.FontColor = 'red';
```

Replace the title for the first slide with the paragraph.

```
contents = find(ppt, 'Title');
replace(contents(1), p);
```

Create a paragraph for the content of the second slide.

```
p1 = Paragraph('My slide content');
append(p1, ' for the second slide');
```

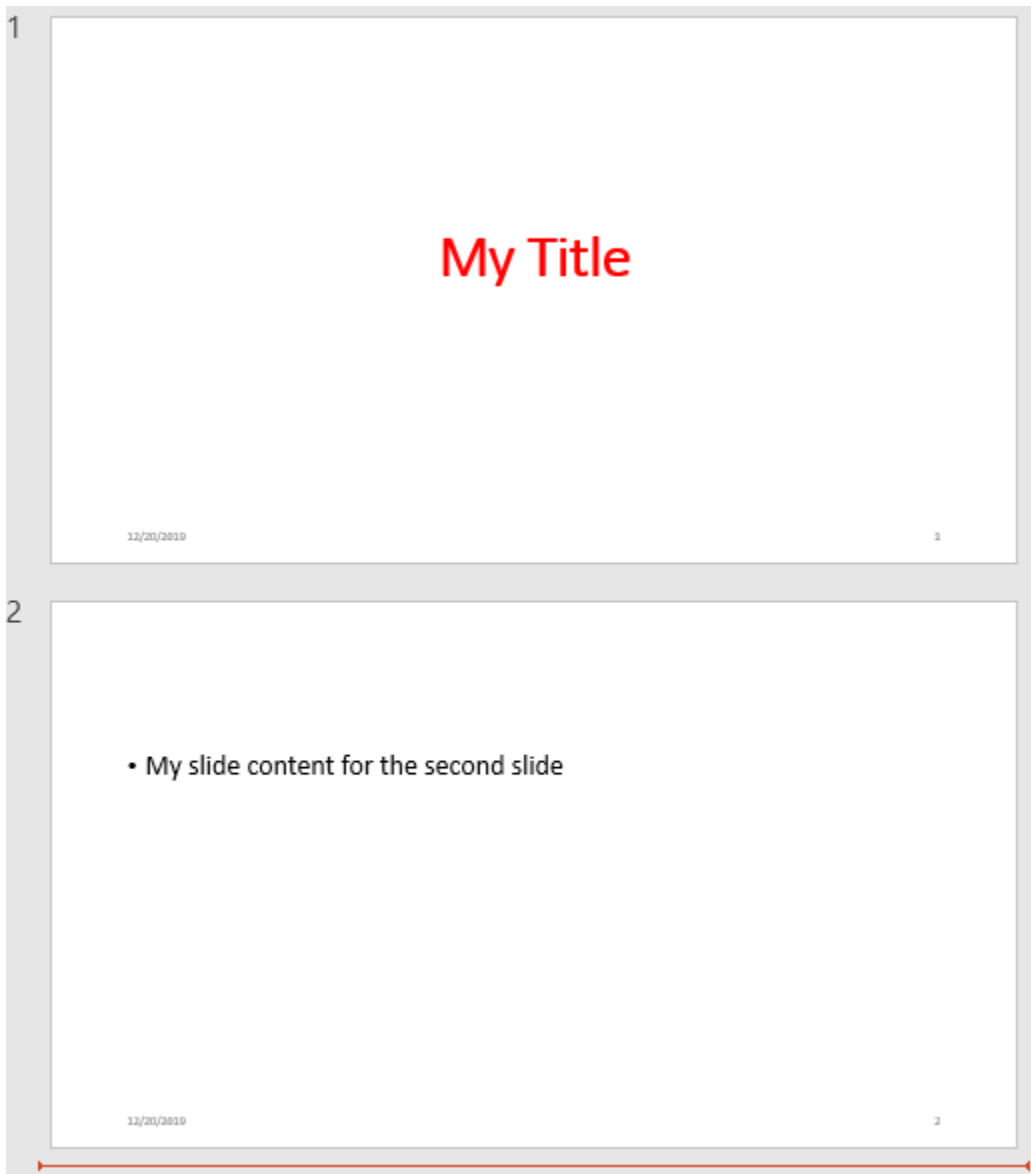
Replace the content with the `p1` paragraph.

```
replace(ppt, 'Content', p1);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here are the slides in the generated presentation:



### See Also

`mreportgen.ppt.Text` | `mreportgen.ppt.TextBox` |  
`mreportgen.ppt.ContentPlaceholder` | `mreportgen.ppt.InternalLink` |  
`mreportgen.ppt.ExternalLink`

### Topics

“Presentation Formatting Approaches” on page 14-18  
“Presentation Format Inheritance” on page 14-21

**Introduced in R2015b**



# mlreportgen.ppt.Picture class

**Package:** mlreportgen.ppt mlreportgen.ppt

Picture to include in presentation

## Description

Use an object of the mlreportgen.ppt.Picture class to include a picture in a presentation.

The mlreportgen.ppt.Picture class is a handle class.

## Class Attributes

ConstructOnLoad	true
HandleCompatible	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

pictureObj = mlreportgen.ppt.Picture() creates an empty Picture object.

pictureObj = mlreportgen.ppt.Picture(picturePath) creates a Picture object that contains the picture specified by picturePath.

---

**Note** The contents of the specified picture file are copied into the output presentation when the presentation is closed. Do not delete or overwrite the picture file before it is copied into the presentation. If you create a picture file and the corresponding mlreportgen.ppt.Picture object in a loop, use a unique file name for the picture file in each loop iteration.

---

## Input Arguments

### picturePath — Path and name of picture file

string scalar | character vector

Path and name of a picture file, specified as a string scalar or character vector. The PPT API supports the image formats in the table.

Image Format	File Extension
Bitmap	.bmp
Enhanced Metafile	.emf
Encapsulated PostScript	.eps
Graphics Interchange Format	.gif

Image Format	File Extension
Joint Photographic Experts Group	.jpeg, .jpg
Portable Network Graphics	.png
Scalable Vector Graphics	.svg
Tagged Image File Format	.tif, .tiff

**Note** To ensure that your version of PowerPoint supports the image formats used in your generated presentation, see File formats that are supported in PowerPoint on the Microsoft website.

## Properties

### Name — Picture name

string scalar | character vector

Picture name, specified as a string scalar or character vector.

#### Attributes:

GetAccess	public
SetAccess	public
NonCopyable	true

### X — Upper-left x-coordinate position of picture

string scalar | character vector

Upper-left x-coordinate position of the picture, specified as a string scalar or character vector that consists of a number followed by an abbreviation for a unit of measurement. For example, "5in" specifies five inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

#### Attributes:

GetAccess	public
SetAccess	public
NonCopyable	true

### Y — Upper-left y-coordinate position of picture

string scalar | character vector

Upper-left y-coordinate position of the picture, specified as a string scalar or character vector that consists of a number followed by an abbreviation for a unit of measurement. For example, "5in" specifies five inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Width — Width of picture**

string scalar | character vector

Width of the picture, specified as a string scalar or character vector that consists of a number followed by an abbreviation for a unit of measurement. For example, "5in" specifies five inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Height — Height of picture**

string scalar | character vector

Height of the picture, specified as a string scalar or character vector that consists of a number followed by an abbreviation for a unit of measurement. For example, "5in" specifies five inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Style — Picture placeholder formatting**

ignored

Picture placeholder formatting. This property is ignored.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Children — Children of this PPT API object**

cell array of PPT API objects

Child elements of this PPT API object, specified as a cell array of PPT API objects.

**Attributes:**

GetAccess	public
SetAccess	private
NonCopyable	true

**Parent — Parent of this PPT API object**

PPT API object

Parent of this PPT API object, specified as a PPT API object.

**Attributes:**

GetAccess	public
SetAccess	private
NonCopyable	true

**Tag — Tag for this PPT API object**

string scalar | character vector

Tag for this PPT API object, specified as a string scalar or character vector. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where issues occur during presentation generation.

**Attributes:**

GetAccess	public
SetAccess	public
NonCopyable	true

**Id — ID for this PPT API object**

string scalar | character vector

ID for this PPT API object, specified as a string scalar or character vector. A session-unique ID is generated as part of the object creation. You can specify an ID to replace the generated ID.

#### Attributes:

GetAccess	public
SetAccess	public
NonCopyable	true

## Methods

### Public Methods

replace	Replace picture with another picture
clone	<p><code>objCopy = clone(obj)</code> returns a copy of the <code>mlreportgen.ppt.Picture</code> object specified by <code>obj</code>.</p> <p>Use the <code>clone</code> method of a <code>Picture</code> object the same way that you use the <code>clone</code> method of a <code>Paragraph</code> object.</p>

## Examples

### Include a Picture in a Presentation

Use an `mlreportgen.ppt.Picture` object to include a picture of an airplane in a presentation.

Create a presentation.

```
import mlreportgen.ppt.*

ppt = Presentation("myPicturePresentation.pptx");
open(ppt);
```

Add a slide with a `Title` and `Content` layout to the presentation.

```
add(ppt, "Title and Content");
```

Create a `Picture` object using an image of an airplane. Specify the size of the picture.

```
plane = Picture(which("b747.jpg"));
plane.Width = "5in";
plane.Height = "2in";
```

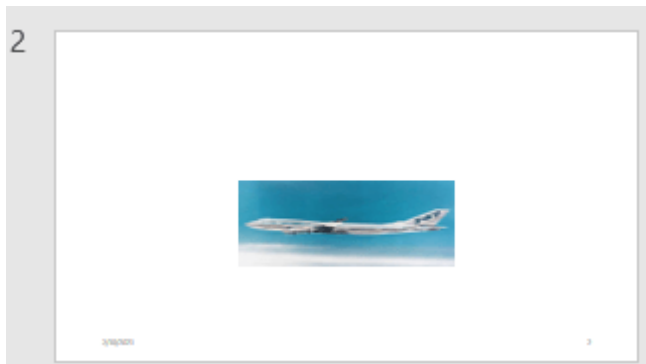
Replace the content of the slide with the picture.

```
replace(ppt, "Content", plane);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here is the generated slide with the image of the airplane:



### Add Figure Snapshot to Presentation

To add an image of a MATLAB figure to a presentation, this example prints the figure to an image file, creates an `mlreportgen.ppt.Picture` object from the file, and adds the `Picture` object to a slide in the presentation.

Create a presentation.

```
import mlreportgen.ppt.*

ppt = Presentation("myFigurePresentation.pptx");
open(ppt);
```

Add a slide to the presentation.

```
slide = add(ppt, "Title and Content");
```

Add a title to the slide.

```
replace(slide, "Title", "surf(peaks)");
```

Create a MATLAB figure with a surface plot.

```
fig = figure;
surf(peaks);
```

Print the figure to an image file. Do not delete the image file before generating the presentation.

```
figSnapshotImage = "figSnapshot.png";
print(fig, "-dpng", figSnapshotImage);
```

Create an `mlreportgen.ppt.Picture` object using the image file.

```
figPicture = Picture(figSnapshotImage);
```

Add the `Picture` object to the slide.

```
replace(slide, "Content", figPicture);
```

Close the presentation.

```
close(ppt);
```

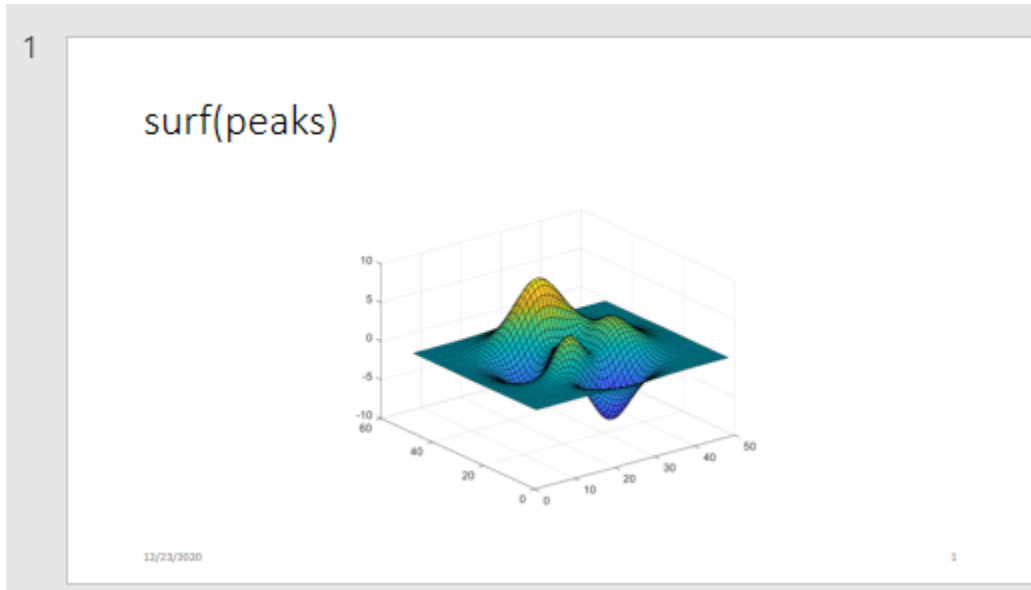
Delete the figure and the image file.

```
delete(fig);  
delete(figSnapshotImage);
```

View the presentation.

```
rptview(ppt);
```

Here is the generated slide with the image of the figure:



## Compatibility Considerations

### SVG support

*Behavior changed in R2021b*

Starting in R2021b, the `mlreportgen.ppt.Picture` class supports Scalable Vector Graphics (SVG) images. To ensure that your version of PowerPoint supports SVG images, see [File formats that are supported in PowerPoint on the Microsoft website](#).

### See Also

`mlreportgen.ppt.PicturePlaceholder` | `mlreportgen.ppt.Presentation`

### Topics

“Create and Format Pictures” on page 14-76

### Introduced in R2015b

## mlreportgen.ppt.PicturePlaceholder class

**Package:** mlreportgen.ppt

Placeholder for slide picture

### Description

An object of the `mlreportgen.ppt.PicturePlaceholder` class represents a picture placeholder in a slide.

The PPT API creates a `PicturePlaceholder` object when you add a slide to a presentation and the slide layout that has a picture placeholder. In the default PPT API, the `Title and Picture` and `Picture with Caption` layouts have picture placeholders.

To find a `PicturePlaceholder` object, use the `find` method of the slide that contains the picture placeholder. To replace the picture placeholder with a picture, use the `replace` method of the `PicturePlaceholder` object. The `replace` method replaces the `PicturePlaceholder` object with an `mlreportgen.ppt.Picture` object.

---

**Note** Microsoft PowerPoint adjusts the size of a picture to fit a picture placeholder. Use a slide with a picture placeholder when you want the placeholder position and size to be fixed in the slide. You can specify the location and size of a picture placeholder by modifying the slide layout in PowerPoint. See “Set Up a PowerPoint Presentation Template” on page 14-23. If the placeholder position and size do not have to be fixed, you can use a slide whose layout has a content placeholder. PowerPoint adjusts the size of a content placeholder to accommodate the size of a picture. See `mlreportgen.ppt.ContentPlaceholder`.

---

The `mlreportgen.ppt.PicturePlaceholder` class is a `handle` class.

### Class Attributes

<code>HandleCompatible</code>	<code>true</code>
<code>ConstructOnLoad</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

### Properties

#### **Bold** — This property is ignored

`[]` (default) | `true` | `false`

This property is ignored.

#### **Font** — This property is ignored

`[]` (default) | character vector | string scalar

This property is ignored.



**ComplexScriptFont — This property is ignored**

[] (default) | true | false

This property is ignored.

**FontColor — This property is ignored**

[] (default) | character vector | string scalar

This property is ignored.

**FontSize — This property is ignored**

[] (default) | character vector | string scalar

This property is ignored.

**Italic — This property is ignored**

[] (default) | true | false

This property is ignored.

**Strike — This property is ignored**

[] (default) | 'single' | 'none' | 'double'

This property is ignored.

**Subscript — This property is ignored**

[] (default) | true | false

This property is ignored.

**Superscript — This property is ignored**

[] (default) | true | false

This property is ignored.

**Underline — This property is ignored**

[] (default) | 'single' | 'double' | ...

This property is ignored.

**BackgroundColor — This property is ignored**

[] (default) | character vector | string scalar

This property is ignored.

**Valign — This property is ignored**

[] (default) | 'top' | 'bottom' | ...

This property is ignored.

**Name — Picture placeholder name**

character vector | string scalar

Picture placeholder name, specified as a character vector or string scalar.

**X — Upper left x-coordinate of position of picture placeholder**

character vector | string scalar

Upper left x-coordinate of the position of the picture placeholder in the slide, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '5in' specifies 5 inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Y — Upper left y-coordinate of position of picture placeholder**

character vector | string scalar

Upper left y-coordinate of the position of the picture placeholder in the slide, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '5in' specifies 5 inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Width — This property is ignored**

character vector | string scalar

This property is ignored.

**Height — This property is ignored**

character vector | string scalar

This property is ignored.

**Style — This property is ignored**

cell array of PPT format objects

This property is ignored.

**Children — Children of this PPT API object**

cell array of PPT objects

Child elements of this object, specified as a cell array of PPT objects. This property is read-only.

**Parent — Parent of this PPT API object**

PPT object

Parent of this object, specified as a PPT object. This property is read-only.

**Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

**Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

**Methods****Public Methods**

replace Replace picture placeholder with picture

**Examples****Replace Picture Placeholder with Picture**

Add a `Title` and `Picture` slide to a presentation and then replace the title and picture placeholders with your own title and picture.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation.

```
ppt = Presentation("myPicturePlaceholderPresentation.pptx");
open(ppt);
```

Add a slide that has a `Title` and `Picture` layout.

```
slide = add(ppt, "Title and Picture");
```

Use the `find` method of the slide object to find the placeholder object that has the name `Title`.

```
titlePlaceholderObj = find(slide, "Title")
```

```
titlePlaceholderObj =
    TextBoxPlaceholder with properties:
```

```
        Bold: []
        Font: []
ComplexScriptFont: []
        FontColor: []
        FontSize: []
```

```
        Italic: []
        Strike: []
        Subscript: []
        Superscript: []
        Underline: []
        BackgroundColor: []
        VAlign: []
        Name: 'Title'
        X: []
        Y: []
        Width: []
        Height: []
        Style: []
        Children: []
        Parent: [1x1 mlreportgen.ppt.Slide]
        Tag: 'ppt.TextBoxPlaceholder:6:9'
        Id: '6:9'
```

The `find` method returns an `mlreportgen.ppt.TextBoxPlaceholder` object.

Replace the placeholder content with the title text.

```
replace(titlePlaceholderObj, "Airplane");
```

Use the `find` method of the slide object to find the placeholder object that has the name `Picture`.

```
picturePlaceholderObj = find(slide, "Picture")
```

```
picturePlaceholderObj =
    PicturePlaceholder with properties:
```

```
        Bold: []
        Font: []
        ComplexScriptFont: []
        FontColor: []
        FontSize: []
        Italic: []
        Strike: []
        Subscript: []
        Superscript: []
        Underline: []
        BackgroundColor: []
        VAlign: []
        Name: 'Picture'
        X: []
        Y: []
        Width: []
        Height: []
        Style: []
        Children: []
        Parent: [1x1 mlreportgen.ppt.Slide]
        Tag: 'ppt.PicturePlaceholder:7:10'
        Id: '7:10'
```

The `find` method returns an `mlreportgen.ppt.PicturePlaceholder` object.

Replace the picture placeholder with a picture.

```
replace(picturePlaceholderObj,Picture("b747.jpg"));
```

Close and view the presentation

```
close(ppt);
rptview(ppt);
```

## Tips

- When you replace a picture placeholder in a presentation with a picture and then use the presentation as a template for a new presentation, the PPT API creates an `mlreportgen.ppt.TemplatePicture` object for the picture in the new presentation. See “Add or Replace a Picture” on page 14-62.
- To see the placeholder objects that the PPT API creates for a slide object, view the `Children` property of the slide. For example, when you add a `Title` and `Picture` slide to a presentation, the `Children` property is an array that contains an `mlreportgen.ppt.TextBoxPlaceholder` object and an `mlreportgen.ppt.PicturePlaceholder` object.

```
ppt = mlreportgen.ppt.Presentation("test.pptx");
open(ppt);
slide = add(ppt,"Title and Picture");
slide.Children(1)
```

ans =

TextBoxPlaceholder with properties:

```

        Bold: []
        Font: []
ComplexScriptFont: []
        FontColor: []
        FontSize: []
        Italic: []
        Strike: []
        Subscript: []
        Superscript: []
        Underline: []
        BackgroundColor: []
        VAlign: []
        Name: 'Title'
        X: []
        Y: []
        Width: []
        Height: []
        Style: []
        Children: []
        Parent: [1x1 mlreportgen.ppt.Slide]
        Tag: 'ppt.TextBoxPlaceholder:6:9'
        Id: '6:9'
```

```
slide.Children(2)
```

ans =

PicturePlaceholder with properties:

```
        Bold: []
        Font: []
ComplexScriptFont: []
        FontColor: []
        FontSize: []
        Italic: []
        Strike: []
        Subscript: []
        Superscript: []
        Underline: []
        BackgroundColor: []
        VAlign: []
        Name: 'Picture'
        X: []
        Y: []
        Width: []
        Height: []
        Style: []
        Children: []
        Parent: [1x1 mlreportgen.ppt.Slide]
        Tag: 'ppt.PicturePlaceholder:7:10'
        Id: '7:10'
```

**See Also**

mlreportgen.ppt.ContentPlaceholder | mlreportgen.ppt.TextBoxPlaceholder |  
mlreportgen.ppt.Picture | mlreportgen.ppt.TemplatePicture |  
mlreportgen.ppt.TablePlaceholder | getLayoutNames

**Topics**

“Access PowerPoint Template Elements” on page 14-32

“Add or Replace a Picture” on page 14-62

**Introduced in R2015b**

# mlreportgen.ppt.Presentation class

**Package:** mlreportgen.ppt

Create a Microsoft PowerPoint presentation container

## Description

Use an object of the mlreportgen.ppt.Presentation class to create a PowerPoint presentation container.

The mlreportgen.ppt.Presentation class is a handle class.

### Class Attributes

ConstructOnLoad	true
HandleCompatible	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

presentationObj = mlreportgen.ppt.Presentation() creates a presentation container and sets the OutputPath property to the default full path for the output presentation file. The default path is the current folder and the default file name is untitled. The TemplatePath property is set to the full path of the default PPT API template.

presentationObj = mlreportgen.ppt.Presentation(outputPath) creates a presentation container with the OutputPath property set to the value of outputPath.

presentationObj = mlreportgen.ppt.Presentation(outputPath,templatePath) also sets the TemplatePath property to the value of templatePath. Use this syntax when you want to use a custom template.

If the outputPath and the templatePath arguments specify the same presentation, the specified presentation becomes the template for the new presentation. When the new presentation is generated, it replaces the original presentation.

## Properties

### TemplatePath — Template file

character vector | string scalar

Template file for this presentation, specified as a character vector or string scalar. Provide the path and file name of the template file, with or without the file name extension. The path can be relative or absolute. If you provide the file name without the path, the file must be in the current folder.

These file extensions are not supported:

- .ppt — PowerPoint 97-2003 Presentation
- .pot — PowerPoint 97-2003 Template
- .pps — PowerPoint 97-2003 Show
- .xml — PowerPoint XML Presentation

Example: "myTemplate.pptx"

Example: "C:\myTemplates\myTemplate.pptx"

### OutputPath — Output presentation file

character vector | string scalar

Output presentation file, specified as a character vector or string scalar. Provide the path and file name of the output presentation file, with or without the file name extension. The path can be relative or absolute. If you provide the file name without the path, the file is created in the current folder.

The file can have one of these extensions:

- .pptx — PowerPoint Presentation
- .pptm — PowerPoint Macro-Enabled Presentation
- .potx — PowerPoint Template
- .potm — PowerPoint Macro-Enabled Template
- .ppsx — PowerPoint Show
- .ppsm — PowerPoint Macro-Enabled Show

The extension of the generated presentation file depends on the extensions specified by the TemplatePath and OutputPath properties. This table shows the supported combinations.

OutputPath extension	TemplatePath extension	Output extension
none	none	.pptx
.pptx	none	.pptx
none	.pptx	.pptx
.pptx	.pptx	.pptx
none	.potx	.pptx
.potx	.potx	.potx
none	.pptm	.pptm
none	.potm	.pptm
.pptm	.pptm	.pptm
none	.ppsx	.ppsx
.ppsx	.ppsx	.ppsx
none	.ppsm	.ppsm
.ppsm	.ppsm	.ppsm

These extensions are not supported for either the template file or the output presentation file:

- .ppt — PowerPoint 97-2003 Presentation



- .pot — PowerPoint 97-2003 Template
- .pps — PowerPoint 97-2003 Slide Show
- .xml — PowerPoint XML Presentation

---

**Note** Specify the `OutputPath` property by providing the `outputPath` argument or set the `OutputPath` property before you open the `mlreportgen.ppt.Presentation` object. When the `Presentation` object is opened, the extension of the file name in the `OutputPath` property is updated, based on the values of the extensions in the `OutputPath` and `TemplatePath` properties as described in the previous table.

---

Example: "myPresentation"

Example: "C:\myPresentations\myPresentation.pptx"

### **Children — Children of this PPT API object**

cell array of PPT objects

Child elements of this object, specified as a cell array of PPT objects. This property is read-only.

### **Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### **Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## **Methods**

### **Public Methods**

open	Open presentation
close	Close presentation
find	Search a presentation
replace	Replace text, tables, or pictures in presentation
add	Add slide to presentation
getMasterNames	Get names of slide masters for presentation
getLayoutNames	Get names of layouts for presentation slide master
getTableStyleNames	Get table style names for presentation
mlreportgen.ppt.Presentation.createTemplate	Create copy of PPT API default presentation template

## **Examples**

### Create Presentation Using Default Formatting

When you create a presentation container and do not specify a template, the template that comes with the PPT API defines the default style of the slides. You can override the default style for individual slides by using format properties and objects.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create an `mlreportgen.ppt.Presentation` object to contain the slides. Do not specify a template. Add a slide for the title and a slide for text.

```
ppt = Presentation('myFirstPresentation.pptx');  
open(ppt);
```

```
titleSlide = add(ppt, 'Title Slide');  
textSlide = add(ppt, 'Title and Content');
```

Specify a title for the presentation. Make the title red by creating the title as an `mlreportgen.ppt.Paragraph` object and setting the `FontColor` format property.

```
paraObj = Paragraph('My First Presentation');  
paraObj.FontColor = 'red';  
replace(titleSlide, 'Title', paraObj);
```

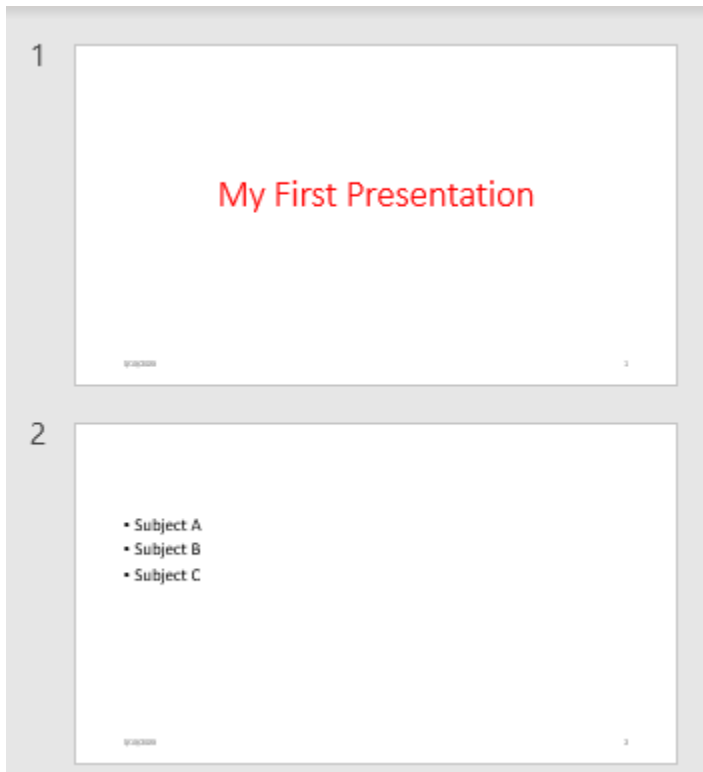
Add content to the second slide.

```
replace(textSlide, 'Content', {'Subject A', 'Subject B', 'Subject C'});
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

Here is the generated presentation:



### Create Presentation and Specify a Template

If the PPT API default template does not meet your formatting requirements, you can specify a different template when you create a presentation container. Use an existing presentation as a template or a customized copy of the default template. This example creates a copy of the default template to illustrate how to specify the template when you create a presentation container.

Create a copy of the PPT API default template named `myTemplate`.

```
mlreportgen.ppt.Presentation.createTemplate('myTemplate.pptx');
```

In PowerPoint, customize the formatting of the slide masters and slide layouts in `myTemplate.pptx`. See “Set Up a PowerPoint Presentation Template” on page 14-23.

Generate a presentation named `myPresentation` using the template `myTemplate`.

```
import mlreportgen.ppt.*
ppt = Presentation('myPresentation.pptx', 'myTemplate.pptx');
open(ppt);
slide = add(ppt, 'Title Slide');
replace(slide, 'Title', 'My Title');
close(ppt);
rptview(ppt);
```

### See Also

`mlreportgen.ppt.Slide` | `mlreportgen.ppt.Paragraph`

**Topics**

“Create a Presentation Object to Hold Content” on page 14-12

“Set Up a PowerPoint Presentation Template” on page 14-23

“Create a Presentation Programmatically” on page 14-49

“Create a Presentation Generator” on page 14-2

**Introduced in R2015a**

# mlreportgen.ppt.ProgressMessage class

**Package:** mlreportgen.ppt

Progress message

## Description

Create a progress message with the specified text originating from the specified source object.

## Construction

`progressMsgObj = ProgressMessage(text, sourcePPTObject)` creates a progress message with the specified text, originating from the specified source object.

## Input Arguments

### **text — Message text**

character vector

The text to display for the message, specified as a character vector.

### **source — PPT object to from which message originates**

PPT object

The PPT object from which the message originates, specified as a PPT object.

## Output Arguments

### **progressMsgObj — Progress message**

mlreportgen.ppt.ProgressMessage object

Progress message, returned as an mlreportgen.ppt.ProgressMessage object.

## Properties

### **Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### **Source — Source PPT object message originates from**

a PPT object

Source PPT object from which the message originates.

### **Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

### Text — Text of message

character vector

Message text, specified as a character vector.

## Methods

Method	Purpose
formatAsHTML	Wrap message in HTML tags.
formatAsText	Format message as text.
passesFilter	Determine if message passes filter.

## Examples

### Create a Progress Message

Create the presentation.

```
import mlreportgen.ppt.*;
pre = Presentation('myPresentation.pptx');
```

Create a message dispatcher.

```
dispatcher = MessageDispatcher.getTheDispatcher;
l = addlistener(dispatcher,'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsText));
```

Dispatch the message.

```
open(pre);
dispatch(dispatcher,ProgressMessage('starting presentation',pre));
```

Add presentation content.

```
titleText = Text('This is a Title');
titleText.Style = {Bold};
```

```
replace(pre,'Title',titleText);
```

Close the presentation and delete the listener.

```
close(pre);
```

Delete the listener to avoid duplicate reporting of message objects during a MATLAB session.

```
delete(l);
```

### **See Also**

dispatch | mlreportgen.ppt.MessageEventData

### **Topics**

“Display Presentation Generation Messages” on page 14-14

**Introduced in R2015b**

## mlreportgen.ppt.RowHeight class

**Package:** mlreportgen.ppt

Table row height

### Description

Use an mlreportgen.ppt.RowHeight format object to specify the height of a table row in a PPT API presentation.

---

**Note** If the row content is too large to fit in the specified height or the specified height value is negative, the generated presentation adjusts the row height to accommodate the content.

---

The mlreportgen.ppt.RowHeight class is a handle class.

### Class Attributes

ConstructOnLoad	true
HandleCompatible	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

rowHeightnObj = mlreportgen.ppt.RowHeight() sets the Value property to '0.41in'.

rowHeightObj = mlreportgen.ppt.RowHeight(value) sets the Value property to value.

### Properties

#### Value — Table row height

'0.41in' (default) | character vector | string scalar

Table row height, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '1in' specifies one inch. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points



**Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

**Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

**Examples****Specify the Height of a Table Row in a Presentation**

To specify the height of a table row in a presentation, add an `mlreportgen.ppt.RowHeight` format object to the `Style` property of the table row. This example generates a table where the height of the first row is two inches and the height of the second row is one inch.

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myRowHeight.pptx");
open(ppt);
slide = add(ppt, "Title and Content");
```

Create a table and specify the height of each row.

```
t = Table(magic(2));
t.row(1).Style = [t.row(1).Style {RowHeight("2in")}];
t.row(2).Height = "1in";
replace(slide, "Content", t);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

**See Also**

`mlreportgen.ppt.Table` | `mlreportgen.ppt.TableRow`

**Topics**

- “Create and Format Tables” on page 14-69
- “Presentation Formatting Approaches” on page 14-18
- “Presentation Format Inheritance” on page 14-21

**Introduced in R2020a**

## mlreportgen.ppt.RowSep class

**Package:** mlreportgen.ppt

Table row separators

### Description

Use an `mlreportgen.ppt.RowSep` format object to customize the lines that separate the rows of a table in a PPT API presentation.

The `mlreportgen.ppt.RowSep` class is a `handle` class.

### Class Attributes

<code>ConstructOnLoad</code>	<code>true</code>
<code>HandleCompatible</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`mlreportgen.ppt.RowSep()` creates an empty `mlreportgen.ppt.RowSep` object. By default, PowerPoint draws thin, solid, black lines between rows. Use the object properties to specify a different line width, style, or color.

`mlreportgen.ppt.RowSep(style)` sets the `Style` property to the value of `style`.

`mlreportgen.ppt.RowSep(style,color)` also sets the `Color` property to the value of `color`.

`mlreportgen.ppt.RowSep(style,color,width)` also sets the `Width` property to the value of `width`.

### Properties

#### Style — Row separator style

`[]` (default) | `'none'` | `'solid'` | `'dot'` | ...

Row separator style, specified as a character vector or string scalar. Specify one of these values:

- `'none'`
- `'solid'`
- `'dot'`
- `'dash'`
- `'largeDash'`
- `'dashDot'`

- 'largeDashDot'
- 'largeDashDotDot'
- 'systemDash'
- 'systemDot'
- 'systemDashDot'
- 'systemDashDotDot'

### Color — Row separator color

[ ] (default) | character vector | string scalar

Row separator color, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

### Width — Row separator width

[ ] (default) | character vector | string scalar

Row separator width, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '3pt' specifies three points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## Examples

## Customize Table Row Separators

To customize the lines that separate table rows, set the properties of an `mlreportgen.ppt.RowSep` object and add the object to the `Style` property of an `mlreportgen.ppt.Table` object. This example makes the row separators dashed and red with a line width of three points.

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myRowSep.pptx");
open(ppt);
```

Add a slide to the presentation.

```
slide = add(ppt, "Title and Table");
```

Create a table and set custom row separators.

```
t = Table(magic(3));
t.Style = [t.Style {RowSep("dash", "red", "3pt")}];
```

Add the title and table to the slide.

```
replace(slide, "Title", "Table with custom row separators");
replace(slide, "Table", t);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here is the table in the generated presentation:

8	1	6
3	5	7
4	9	2

12/6/2019 1

## See Also

`mlreportgen.ppt.Table` | `mlreportgen.ppt.ColSep` | `mlreportgen.ppt.Border`

**Topics**

“Create and Format Tables” on page 14-69

“Presentation Formatting Approaches” on page 14-18

“Presentation Format Inheritance” on page 14-21

**Introduced in R2020a**

## mlreportgen.ppt.Slide class

**Package:** mlreportgen.ppt

Presentation slide

### Description

An object of the `mlreportgen.ppt.Slide` class represents a slide in a Microsoft PowerPoint presentation. To create a `Slide` object and add it to a presentation, use the `add` method of an `mlreportgen.ppt.Presentation` object. The `add` method returns the `Slide` object. You can use the `Slide` object methods to add, find, and replace slide content.

The `mlreportgen.ppt.Slide` class is a handle class.

### Class Attributes

Abstract	true
ConstructOnLoad	true
HandleCompatible	true

For information on class attributes, see “Class Attributes”.

### Properties

#### Layout — Slide layout name

character vector

Slide layout name, specified as a character vector. This property is read-only.

#### SlideMaster — Slide master name

character vector

Slide master name, specified as a character vector. This property is read-only.

#### Name — Slide name

' ' (default) | character vector | string scalar

Slide name, specified as a character vector or string scalar. You can set the `Name` property to identify a slide in a presentation. See the `find` method of the `mlreportgen.ppt.Presentation` class.

#### Style — Slide formatting

[] | cell array of PPT format objects

Slide formatting, specified as a cell array of PPT format objects.

Add format objects by concatenating the existing value of the `Style` property with a cell array that contains the format objects that you are adding. For example:

```
import mlreportgen.ppt.*
ppt = Presentation('MyPresentation');
open(ppt);
```

```
slide = add(ppt, 'Title Only');
slide.Style = [slide.Style {Bold(true), FontColor('red')}]};
replace(slide, 'Title', 'My Title is Bold and Red');
close(ppt);
rptview(ppt);
```

See “Presentation Formatting Approaches” on page 14-18.

### Children — Children of this slide

cell array of PPT objects

Child elements of this slide, specified as a cell array of PPT objects. This property is read-only.

### Parent — Parent of this slide object

PPT object

Parent of this slide, specified as a PPT object. This property is read-only.

### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## Methods

### Public Methods

find      Search slide for content  
 add      Add text box, table, or picture to slide  
 replace   Replace text, tables, or pictures in a slide

## Examples

### Add a Slide to a Presentation

To add a slide to a presentation, use the add method of the mlreportgen.ppt.Presentation object. The add method returns an mlreportgen.ppt.Slide object. Use the replace method of the Slide object to replace content placeholders with your content.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create and open a presentation.

```
ppt = Presentation("myPresentation.pptx");
open(ppt);
```

Add a slide that has a Title and Table layout.

```
slide = add(ppt, "Title and Table");
```

Replace the Title placeholder in the slide with the title text.

```
replace(slide, "Title", "magic(4)");
```

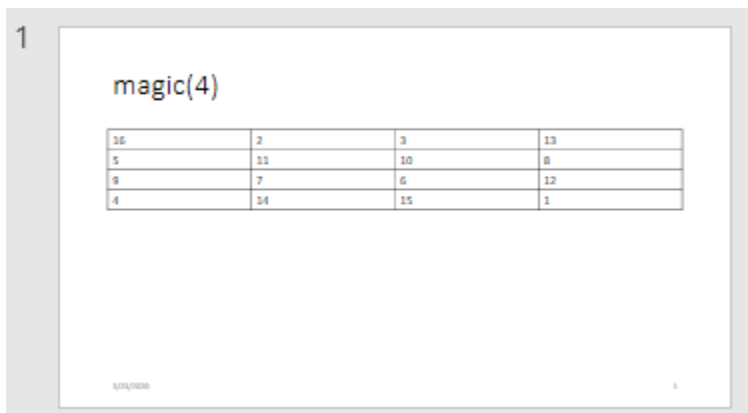
Replace the Table placeholder in the slide with a table.

```
tbl = Table(magic(4));
replace(slide, "Table", tbl);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here is the generated presentation:



## See Also

`mlreportgen.ppt.Presentation`

## Topics

“Create a Presentation Programmatically” on page 14-49

“Add Slides” on page 14-56

“Add and Replace Presentation Content” on page 14-58

**Introduced in R2015a**



# mlreportgen.ppt.Strike class

**Package:** mlreportgen.ppt

Strikethrough text

## Description

Format that draws a line through presentation text.

The mlreportgen.ppt.Strike class is a handle class.

## Class Attributes

HandleCompatible	true
ConstructOnLoad	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`strikeObj = mlreportgen.ppt.Strike()` draws a single, horizontal line through text.

`strikeObj = mlreportgen.ppt.Strike(style)` sets the `Style` property to the specified strike style.

## Properties

### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

### Style — Strikethrough style

'single' (default) | 'none' | 'double'

Strikethrough style, specified as one of these values:

- 'single' — Single horizontal line
- 'none' — No strikethrough line
- 'double' — Double horizontal line

## Examples

### Draw a Double Line Through Presentation Text

Create a presentation.

```
import mlreportgen.ppt.*  
  
ppt = Presentation('myStrikePresentation.pptx');  
open(ppt);  
slide = add(ppt, 'Title and Content');
```

Create a paragraph and append text with double strikethrough formatting.

```
p = Paragraph('Hello World');  
  
tStrike = Text(' strikethrough text');  
tStrike.Style = {Strike('double')};  
append(p, tStrike);
```

Create text without strikethrough formatting and append it to the paragraph.

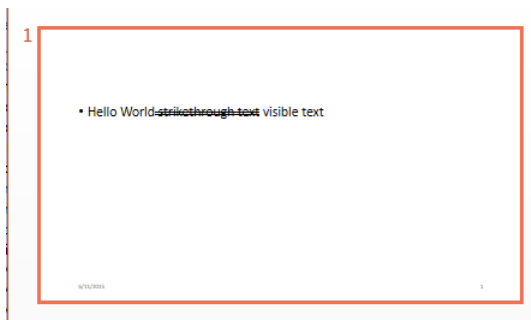
```
tVisible = Text(' visible text');  
append(p, tVisible);
```

Add the paragraph to the slide.

```
replace(slide, 'Content', p);
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```



### See Also

mlreportgen.ppt.Underline | mlreportgen.ppt.Bold | mlreportgen.ppt.Italic |  
mlreportgen.ppt.FontColor | mlreportgen.ppt.FontFamily |  
mlreportgen.ppt.FontSize

**Topics**

“Create and Format Text” on page 14-65

“Presentation Formatting Approaches” on page 14-18

**Introduced in R2015b**

## mlreportgen.ppt.Subscript class

**Package:** mlreportgen.ppt

Subscript text

### Description

Format that renders text in a presentation as a subscript.

The mlreportgen.ppt.Subscript class is a handle class.

### Class Attributes

HandleCompatible	true
ConstructOnLoad	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

subscriptObj = mlreportgen.ppt.Subscript() creates a subscript object.

subscriptObj = mlreportgen.ppt.Subscript(value) creates a subscript object and sets the Value property to value.

## Properties

### Value — Option to display text as subscript

true or 1 (default) | false or 0

Option to display text as a subscript, specified as a numeric or logical 1 (true) or 0 (false). A setting of true or 1 renders text as a subscript.

### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## Examples

### Render Presentation Text as a Subscript

Set up a paragraph to display H<sub>2</sub>O.

Set up a presentation.

```
import mlreportgen.ppt.*  
  
ppt = Presentation('mySubscript.pptx');  
open(ppt);  
slide = add(ppt, 'Title and Content');
```

Create the subscript text and append it to a paragraph with regular text.

```
sub = Text('2');  
sub.Style = {Subscript(true)};  
  
para = Paragraph('H');  
append(para, sub);  
append(para, 'O');
```

Replace the slide content with the paragraph.

```
replace(slide, 'Content', para);
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

### See Also

mlreportgen.ppt.Superscript

### Topics

“Create and Format Text” on page 14-65

“Presentation Formatting Approaches” on page 14-18

**Introduced in R2015b**

## mlreportgen.ppt.Superscript class

**Package:** mlreportgen.ppt

Superscript text

### Description

Format that renders text in a presentation as a superscript.

The mlreportgen.ppt.Superscript class is a handle class.

### Class Attributes

HandleCompatible	true
ConstructOnLoad	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

superscriptObj = mlreportgen.ppt.Superscript() creates a superscript object.

superscriptObj = mlreportgen.ppt.Superscript(value) creates a superscript object and sets the Value property to value.

## Properties

### Value — Option to display text as superscript

true or 1 (default) | false or 0

Option to display text as a superscript, specified as a numeric or logical 1 (true) or 0 (false). A setting of true or 1 renders text as a superscript.

### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## Examples

### Render Presentation Text as a Superscript

Set up a paragraph to display  $x^2$ .

Set up a presentation.

```
import mlreportgen.ppt.*  
  
ppt = Presentation('mySuperscript.pptx');  
open(ppt);  
slide = add(ppt, 'Title and Content');
```

Create the superscript text and append it to a paragraph with regular text.

```
super = Text('2');  
super.Style = {Superscript(true)};  
  
para = Paragraph('x');  
append(para, super);
```

Replace the slide content with the paragraph.

```
replace(slide, 'Content', para);
```

Close and view presentation.

```
close(ppt);  
rptview(ppt);
```

### See Also

mlreportgen.ppt.Subscript

### Topics

“Create and Format Text” on page 14-65

“Presentation Formatting Approaches” on page 14-18

### Introduced in R2015b

## mlreportgen.ppt.Table class

**Package:** mlreportgen.ppt mlreportgen.ppt

Table in presentation

### Description

Use an object of the `mlreportgen.ppt.Table` class to include a table in a PPT API presentation.

Create a table using one of these approaches:

- Create an empty table and append table rows that have table entries for each column.
- Create a table from an array or cell array that specifies the table content.

After you create a table, you can add rows to the table, and add entries to each table row.

The `mlreportgen.ppt.Table` class is a `handle` class.

### Class Attributes

<code>HandleCompatible</code>	<code>true</code>
<code>ConstructOnLoad</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`tableObj = mlreportgen.ppt.Table()` creates an empty `Table` object.

`tableObj = mlreportgen.ppt.Table(nCols)` sets the `NCols` property to `nCols`.

`tableObj = mlreportgen.ppt.Table(tableValues)` returns a table that has the content specified by `tableValues`.

`tableObj = mlreportgen.ppt.Table(tableValues, styleName)` returns a table that has the specified content and sets the `StyleName` property to `styleName`. To get the list of valid style names, use the `getTableStyleNames` method of the `mlreportgen.ppt.Presentation` object.

### Input Arguments

#### **tableValues — Table values**

two-dimensional numeric array | two-dimensional categorical array | two-dimensional cell array

Table values, specified as a two-dimensional numeric array, categorical array, or cell array of numbers, character vectors, string scalars, or `mlreportgen.ppt.Paragraph` objects.



## Properties

### **NColumns — Number of table columns**

0 (default) | integer

Number of table columns, specified as an integer. This property is read-only. To specify the number of columns, create a table by using the syntax `mlreportgen.ppt.Table(nColumns)`. Otherwise, the `Table` constructor determines the number of columns from the table content. If you add rows to a table or entries to a row and the number of columns changes, the `NColumns` property value is updated. If the rows do not all have the same number of entries, the row with the largest number of table entries determines the number of columns in the table.

### **StyleName — Table style name**

[] (default) | character vector | string scalar

Table style name, specified as a character vector or string scalar. To get the list of valid style names, use the `getTableStyleNames` method of the `mlreportgen.ppt.Presentation` object.

### **BackgroundColor — Table background color**

[] (default) | character vector | string scalar

Table background color, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

### **ColSpecs — Table column format objects**

[] (default) | array of `mlreportgen.ppt.ColSpec` objects

Array of `mlreportgen.ppt.ColSpec` objects that specify the width, alignment, and other formatting properties of the table columns. The first object applies to the first column, the second object applies to the second column, and so on.

### **FlowDirection — Table column flow direction**

[] (default) | 'LeftToRight' | 'RightToLeft'

Table column flow direction, specified as a character vector or string scalar. Specify one of these values:

- 'LeftToRight' — Left-to-right column order
- 'RightToLeft' — Right-to-left column order

See “Order Table Columns” on page 12-696.

### **Border — Border style**

'solid' (default) | 'none' | 'dot' | ...

Border style, specified as a character vector or string scalar. Specify one of these values:

- 'none'
- 'solid'

- 'dot'
- 'dash'
- 'largeDash'
- 'dashDot'
- 'largeDashDot'
- 'largeDashDotDot'
- 'systemDash'
- 'systemDot'
- 'systemDashDot'
- 'systemDashDotDot'

**BorderColor — Border color**

'black' (default) | character vector | string scalar

Border color, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

**BorderWidth — Border width**

'1pt' (default) | character vector | string scalar

Border width, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '3pt' specifies three points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**ColSep — Column separator style**

'solid' (default) | 'none' | 'dot' | ...

Column separator style, specified as a character vector or string scalar. Specify one of these values:

- 'none'
- 'solid'
- 'dot'
- 'dash'
- 'largeDash'
- 'dashDot'

- 'largeDashDot'
- 'largeDashDotDot'
- 'systemDash'
- 'systemDot'
- 'systemDashDot'
- 'systemDashDotDot'

### **ColSepColor — Column separator color**

'black' (default) | character vector | string scalar

Column separator color, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

### **ColSepWidth — Column separator width**

'1pt' (default) | character vector | string scalar

Column separator width, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '3pt' specifies three points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

### **RowSep — Row separator style**

'solid' (default) | 'none' | 'dot' | ...

Row separator style, specified as a character vector or string scalar. Specify one of these values:

- 'none'
- 'solid'
- 'dot'
- 'dash'
- 'largeDash'
- 'dashDot'
- 'largeDashDot'
- 'largeDashDotDot'
- 'systemDash'
- 'systemDot'

- 'systemDashDot'
- 'systemDashDotDot'

**RowSepColor — Row separator color**

'black' (default) | character vector | string scalar

Row separator color, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

**RowSepWidth — Row separator width**

'1pt' (default) | character vector | string scalar

Row separator width, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '3pt' specifies three points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Font — Font family for text in this table**

[] (default) | character vector | string scalar

Font family for the text in this table, specified as a character vector or string scalar. Specify a font that appears in the font list in Microsoft PowerPoint. To see the font list, in PowerPoint, on the **Home** tab, in the **Font** group, click the arrow to the right of the font.

**ComplexScriptFont — Font family for complex scripts**

[] (default) | character vector | string scalar

Font family for complex scripts, specified as a character vector or string scalar. Specify a font family to use when substituting in a locale that requires a complex script, such as Arabic or Asian, for rendering text.

**FontColor — Font color for text in this table**

[] (default) | character vector | string scalar

Font color for text in this table, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

**FontSize — Font size of text in this table**

[] (default) | character vector | string scalar

Font size of text in this table, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '11pt' specifies 11 points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

#### **Name — Table name**

' ' (default) | character vector | string scalar

Table name, specified as a character vector or string scalar.

#### **X — Upper left x-coordinate of position of table**

[] (default) | character vector | string scalar

Upper left x-coordinate of the position of the table, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '5in' specifies 5 inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

#### **Y — Upper left y-coordinate of position of table**

[] (default) | character vector | string scalar

Upper left y-coordinate of the position of the table, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '5in' specifies 5 inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

#### **Width — Width of table**

[] (default) | character vector | string scalar

Width of table, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '5in' specifies five inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Height — Height of table**

[ ] (default) | character vector | string scalar

Height of table, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '5in' specifies five inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

If the table height is specified and the row height is not specified for any row, the height of all rows is the same. The row height is determined by dividing the table height by the number of rows. If the height is specified for at least one row, the PPT API ignores the table height. Microsoft PowerPoint determines the height of the rows for which the height is not specified.

**Style — Table formatting**

cell array of PPT format objects

Table formatting, specified as a cell array of PPT format objects. Formats that do not apply to a `Table` object are ignored.

**Children — Children of this PPT API object**

cell array of PPT objects

Child elements of this object, specified as a cell array of PPT objects. This property is read-only.

**Parent — Parent of this PPT API object**

PPT object

Parent of this object, specified as a PPT object. This property is read-only.

**Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

**Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

**Methods****Public Methods**

append	Append row to table
replace	Replace table with another table
row	Access table row
entry	Access table entry
clone	<p><code>objCopy = clone(obj)</code> returns a copy of the <code>mlreportgen.ppt.Table</code> object specified by <code>obj</code>.</p> <p>Use the <code>clone</code> method of a <code>Table</code> object the same way that you use the <code>clone</code> method of a <code>Paragraph</code> object.</p>

**Examples****Add a Table to a Presentation**

Use `mlreportgen.ppt.Table` objects to include tables in a presentation.

Create a presentation.

```
import mlreportgen.ppt.*

ppt = Presentation('myTablePresentation.pptx');
open(ppt);

slide1 = add(ppt, 'Title and Table');
slide2 = add(ppt, 'Title and Table');
```

Create a table from a cell array.

```
table1 = Table({'a', 'b', 'c', 'd'});
table1.Children(1).FontColor = 'red';
table1.Children(2).FontColor = 'green';
```

Use the `find` method to find the slides that have objects with a `Name` property set to `Table`. The default PPT API `Title` and `Table` layout slide has a `Table` object.

```
contents = find(ppt, 'Table');
```

Replace the contents of the first slide with `table1`.

```
replace(contents(1), table1);
```

Create a second table from the output of the MATLAB® `magic` function.

```
table2 = Table(magic(9));
```

Replace the contents of the second slide with `table2`.

```
replace(contents(2), table2);
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

Here are the slides in the generated presentation:

The first slide displays a 2x2 table with the following structure:

a	b
c	d

The second slide displays a 9x9 magic square table with the following data:

47	58	69	80	1	12	23	34	45
57	68	79	9	11	22	33	44	46
67	78	8	10	21	32	43	54	56
77	7	18	20	31	42	53	55	66
6	17	19	30	41	52	63	65	76
16	27	29	40	51	62	64	75	5
26	28	39	50	61	72	74	4	15
36	38	49	60	71	73	3	14	25
37	48	59	70	81	2	13	24	35



## Order Table Columns

To specify the order of the columns of a table in a presentation, use the `FlowDirection` property of the `Table` object.

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation('myFlowDirection.pptx');
open(ppt);
```

Add a slide to a presentation.

```
slide = add(ppt, 'Title and Content');
```

Create a table and specify the table flow direction.

```
t = Table({'entry(1,1)', 'entry(1,2)'; 'entry(2,1)', 'entry(2,2)'});
t.FlowDirection = 'RightToLeft';
```

Add the slide to the table.

```
replace(slide, 'Content', t);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here is the slide in the generated presentation:

entry(1,2)	entry(1,1)
entry(2,2)	entry(2,1)

## See Also

`mlreportgen.ppt.TableRow` | `mlreportgen.ppt.TableEntry` |  
`mlreportgen.ppt.TablePlaceholder` | `mlreportgen.ppt.TemplateTable`

**Topics**

“Create and Format Tables” on page 14-69

**Introduced in R2015b**

# mlreportgen.ppt.TableEntry class

**Package:** mlreportgen.ppt mlreportgen.ppt

Table entry

## Description

Use an object of the `mlreportgen.ppt.TableEntry` class to create a table entry to add to a row of a table in a PPT API presentation.

To add content to a table entry, append a character vector or one or more `mlreportgen.ppt.Paragraph` objects to the `TableEntry` object.

The `mlreportgen.ppt.TableEntry` class is a handle class.

## Class Attributes

<code>HandleCompatible</code>	<code>true</code>
<code>ConstructOnLoad</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`tableEntryObj = mlreportgen.ppt.TableEntry()` creates an empty `TableEntry` object.

## Properties

### Font — Font family for text in this table entry

[ ] (default) | character vector | string scalar

Font family for the text in this table entry, specified as a character vector or string scalar. Specify a font that appears in the font list in Microsoft PowerPoint. To see the font list, in PowerPoint, on the **Home** tab, in the **Font** group, click the arrow to the right of the font.

### ComplexScriptFont — Font family for complex scripts

[ ] (default) | character vector | string scalar

Font family for complex scripts, specified as a character vector or string scalar. Specify a font family to use when substituting in a locale that requires a complex script, such as Arabic or Asian, for rendering text.

### FontColor — Font color for text in this table entry

[ ] (default) | character vector | string scalar

Font color for text in this table entry, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

### FontSize — Font size of text in this table entry

[] (default) | character vector | string scalar

Font size of text in this table entry, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '11pt' specifies 11 points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

### BackgroundColor — Table entry background color

[] (default) | character vector | string scalar

Table entry background color, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

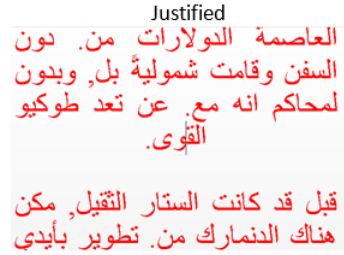

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

### HALign — Horizontal alignment of table entry content

[] (default) | 'center' | 'left' | ...

Horizontal alignment of table entry content, specified as a character vector or string scalar. Specify one of the values in the table.

Value	Description
'center'	Centered
'left'	Left-justified
'right'	Right-justified
'justified'	Left and right-justified, spacing words evenly
'distributed'	Left and right-justified, spacing letters evenly
'thaiDistributed'	Left and right-justified Thai text, spacing characters evenly

Value	Description
'justifiedLow'	Justification for Arabic text <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;"> <p>Justified</p>  </div> <div style="text-align: center;"> <p>Justified Low</p>  </div> </div>

### VALign – Vertical alignment of table entry content

[] (default) | 'top' | 'bottom' | ...

Vertical alignment of table entry content, specified as a character vector or string scalar. Specify one of the values in the table.

Value	Description
'top'	Vertically aligned to the top
'bottom'	Vertically aligned to the bottom of the table entry
'middle'	Vertically aligned to the middle of the table entry
'topCentered'	Vertically aligned to the top and horizontally aligned to the center of the table entry
'bottomCentered'	Vertically aligned to the bottom and horizontally aligned to the center of the table entry
'middleCentered'	Vertically aligned to the middle and horizontally aligned to the center of the table entry

### TextOrientation – Orientation of table entry text

[] (default) | 'horizontal' | 'down' | 'up'

Orientation of the table entry text, specified as a character vector or string scalar. Specify one of the values in the table.

Value	Description
'horizontal'	Text orientation is horizontal.
'down'	Text orientation is vertical, with the content rotated 90 degrees, clockwise.
'up'	Text orientation is vertical, with the content rotated 90 degrees, counterclockwise.

### ColSpan – Number of columns spanned

1 (default) | double

Number of columns spanned by this table entry, specified as a double.

To create a table that has an entry that spans multiple columns, build the table from `mlreportgen.ppt.TableEntry` and `mlreportgen.ppt.TableRow` objects. See “Create Table

Entries That Span Multiple Columns” on page 12-706. In the row that contains the entry that spans columns, define the entry for the first column of the span. Do not define entries for the other columns in the span.

An entry that spans columns shares a border with the entry that is in the row below and in the first column of the span. The bottom border of the spanning entry determines the style of the shared border. In the row that contains the spanning entry, the entries for the second through last columns of the span are undefined. The top borders of the corresponding entries in the row below determine the borders under the undefined entries. To specify the border under the spanning entry for the entire column span:

- Specify the bottom border of the entry that spans the columns.
- In the next row, specify the top borders of the entries in the second through last columns of the span.

For an example, see “Customize Bottom Border of Entry That Spans Multiple Columns” on page 12-709.

### **RowSpan — Number of rows spanned**

1 (default) | double

Number of rows spanned by this table entry, specified as a double.

To create a table that has an entry that spans multiple rows, build the table from `mlreportgen.ppt.TableEntry` and `mlreportgen.ppt.TableRow` objects. See “Create Table Entries That Span Multiple Rows” on page 12-707. In the column that contains the entry that spans rows, define the entry for the first row of the span. Do not define entries for the other rows in the span.

An entry that spans rows shares a border with the entry that is in the column to the right and in the first row of the span. The right border of the spanning entry determines the style of the shared border. In the column that contains the spanning entry, the entries for the second through last rows of the span are undefined. The left borders of the corresponding entries in the column to the right determine the right borders of the undefined entries. To specify the right border of the spanning entry for the entire row span:

- Specify the right border of the entry that spans the rows.
- In the next column, specify the left borders of the entries in the second through last rows of the span.

For an example, see “Customize Right Border of Entry That Spans Multiple Rows” on page 12-711.

### **Border — Table entry border style**

[] (default) | 'solid' | 'none' | 'dot' | ...

Table entry border style, specified as a character vector or string scalar. Specify one of these values:

- 'none'
- 'solid'
- 'dot'
- 'dash'
- 'largeDash'

- 'dashDot'
- 'largeDashDot'
- 'largeDashDotDot'
- 'systemDash'
- 'systemDot'
- 'systemDashDot'
- 'systemDashDotDot'

A conflict occurs if a border segment is shared by two table entries. For a conflicting horizontal border segment, PowerPoint ignores the formats specified by the entry on the bottom. For a conflicting vertical border segment, PowerPoint ignores the formats specified by the entry on the right.

### **BorderColor** — Table entry border color

[ ] (default) | character vector | string scalar

Table entry border color, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

### **BorderWidth** — Table entry border width

[ ] (default) | character vector | string scalar

Table entry border width, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '3pt' specifies three points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

### **Style** — Table entry formatting

cell array of PPT format objects

Table entry formatting, specified as a cell array of PPT format objects. Formats that do not apply to a TableEntry object are ignored.

### **Children** — Children of this PPT API object

cell array of PPT objects

Child elements of this object, specified as a cell array of PPT objects. This property is read-only.

### **Parent** — Parent of this PPT API object

PPT object

Parent of this object, specified as a PPT object. This property is read-only.

### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## Methods

### Public Methods

append	Append text or paragraph to table entry
clone	<p><code>objCopy = clone(obj)</code> returns a copy of the <code>mlreportgen.ppt.TableEntry</code> object specified by <code>obj</code>.</p> <p>Use the <code>clone</code> method of a <code>TableEntry</code> object the same way that you use the <code>clone</code> method of a <code>Paragraph</code> object.</p>

## Examples

### Create a Table Using Entries and Rows

Create a table for a presentation by adding entries to rows and the rows to a table.

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation('myTableRowPresentation.pptx');
open(ppt);
```

Add a slide to the presentation.

```
add(ppt, 'Title and Content');
```

Create a table.

```
table = Table();
```

Create the first table row.



```
tr1 = TableRow();
tr1.Style = [tr1.Style {Bold(true)}];
```

Create the table entries for the first row.

```
te1tr1 = TableEntry();
p = Paragraph('first entry');
p.FontColor = 'red';
append(te1tr1,p);
append(tr1,te1tr1);

te2tr1 = TableEntry();
append(te2tr1,'second entry');
append(tr1,te2tr1);

te3tr1 = TableEntry();
te3tr1.FontColor = 'green';
append(te3tr1,'third entry');
append(tr1,te3tr1);
```

Append the first table row to the table.

```
append(table,tr1);
```

Create the second table row, append the table entries to the row, and append the row to the table.

```
tr2 = TableRow();
te1tr2 = TableEntry();
te1tr2.FontColor = 'red';
p = Paragraph('first entry');
append(te1tr2,p);
append(tr2,te1tr2);

te2tr2 = TableEntry();
append(te2tr2,'second entry');
append(tr2,te2tr2);

te3tr2 = TableEntry();
te3tr2.FontColor = 'green';
append(te3tr2,'third entry');
append(tr2,te3tr2);
append(table,tr2);
```

Add the table to the presentation.

```
contents = find(ppt,'Content');
replace(contents(1),table);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here is the table in the generated presentation:

first entry	second entry	third entry
first entry	second entry	third entry

12/6/2019 1

### Create Table Entries That Span Multiple Columns

Use the `ColSpan` property to specify that a table entry spans multiple columns.

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myColSpan.pptx");
open(ppt);
```

Add a slide to the presentation.

```
slide = add(ppt, "Title and Table");
```

Create a table that has four columns.

```
t = Table(4);
t.StyleName = "Medium Style 2 - Accent 1";
t.Style = {VAlign("middleCentered")};
```

Create the header row. This row has a single table entry that spans all four columns.

```
tr1 = TableRow();
tr1te1 = TableEntry("Header Row");
tr1te1.ColSpan = 4;
append(tr1, tr1te1);
append(t, tr1);
```

Create the subheader row. This row has two table entries, each of which spans two columns.

```
tr2 = TableRow();
tr2te1 = TableEntry("Subheader 1");
tr2te1.ColSpan = 2;
append(tr2, tr2te1);
tr2te2 = TableEntry("Subheader 2");
```

```
tr2te2.ColSpan = 2;
append(tr2, tr2te2);
append(t, tr2);
```

Create the content row. This row has four table entries each of which spans a single column.

```
tr3 = TableRow();
append(tr3, TableEntry("entry(1,1)"));
append(tr3, TableEntry("entry(1,2)"));
append(tr3, TableEntry("entry(1,3)"));
append(tr3, TableEntry("entry(1,4)"));
append(t, tr3);
```

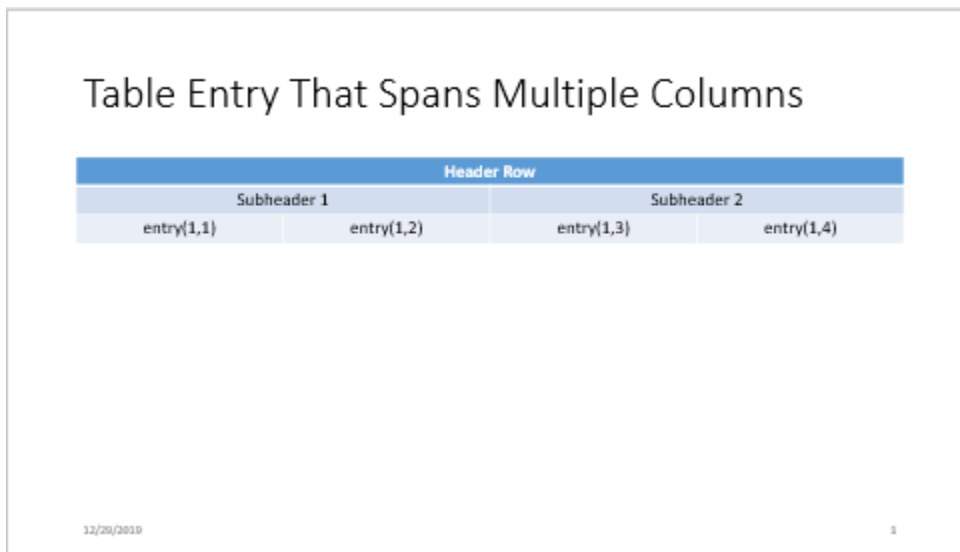
Add a title and the table to the slide.

```
replace(slide, "Title", "Table Entry That Spans Multiple Columns");
replace(slide, "Table", t);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here is the slide in the generated presentation:



The slide displays a table with the following structure:

Header Row			
Subheader 1	Subheader 2		
entry(1,1)	entry(1,2)	entry(1,3)	entry(1,4)

The slide also includes a date '12/20/2019' in the bottom left corner and a page number '1' in the bottom right corner.

### Create Table Entries That Span Multiple Rows

Use the RowSpan property to specify that a table entry spans multiple rows.

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myRowSpan.pptx");
open(ppt);
```

Add a slide to the presentation.

```
slide = add(ppt, "Title and Table");
```

Create a table that has three columns.

```
t = Table(3);  
t.Style = [t.Style {VAlign("middleCenter")}];
```

Create the first table row.

```
tr1 = TableRow();
```

Create an entry in the first column that spans all four rows.

```
tr1te1 = TableEntry("Header Column");  
tr1te1.RowSpan = 4;  
append(tr1, tr1te1);
```

Create an entry in the second column that spans the first two rows.

```
tr1te2 = TableEntry("Subheader 1");  
tr1te2.RowSpan = 2;  
append(tr1, tr1te2);
```

Create an entry in the third column that spans one row.

```
append(tr1, TableEntry("entry(1,1)"));  
append(t, tr1);
```

Create the second table row. Create the third column entry. The first and second column entries are already defined because the first and second column entries span the first two rows.

```
tr2 = TableRow();  
append(tr2, TableEntry("entry(2,1)"));  
append(t, tr2);
```

Create the third table row and create the entries for the second and third columns. Make the second column entry span two rows. The first column entry is already defined because the row one, column one entry spans all of the rows.

```
tr3 = TableRow();  
tr3te2 = TableEntry("Subheader 2");  
tr3te2.RowSpan = 2;  
append(tr3, tr3te2);  
append(tr3, TableEntry("entry(3,1)"));  
append(t, tr3);
```

Create the fourth table row. Create the entry for only the third column. The first and second column entries are already defined by previous rows that span multiple columns.

```
tr4 = TableRow();  
append(tr4, TableEntry("entry(4,1)"));  
append(t, tr4);
```

Add a title and the table to the slide.

```
replace(slide, "Title", "Table Entry That Spans Multiple Rows");  
replace(slide, "Table", t);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here is the slide in the generated presentation:

**Table Entry That Spans Multiple Rows**

Header Column	Subheader 1	entry(1,1)
		entry(2,1)
	Subheader 2	entry(3,1)
		entry(4,1)

12/20/2019 1

### Customize Bottom Border of Entry That Spans Multiple Columns

To customize the bottom border of an entry that spans multiple columns:

- Specify the bottom border of the entry that spans the columns. This specification determines the border that is shared with the entry in the next row that is in the first column of the span.
- In the next row, specify the top borders of the entries that are in the remaining columns of the span.

This example defines a table that has two rows and three columns. The entry in row one and column one spans three columns. To make the bottom border of this entry red along the entire span, the example specifies that:

- The bottom border of the row one, column one entry is red.
- The top borders of the row two, column two and row two, column three entries are red.

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myColSpanBorder.pptx");
open(ppt);
```

Add a slide to the presentation.

```
slide = add(ppt, "Title and Table");
```

Create a table that has three columns.

```
t = Table(3);  
t.Style = [t.Style {VAlign("middleCenter")}];
```

Create the first row of the table.

```
tr1 = TableRow();
```

Create the first entry of the first row and specify that the entry spans three columns. Specify that the bottom border of the entry is red. Append the entry to the row.

```
te11 = TableEntry("entry 1,1");  
te11.ColSpan = 3;  
border = Border();  
border.BottomColor = 'red';  
te11.Style = [te11.Style {border}];  
append(tr1,te11);
```

Create the second row of the table.

```
tr2 = TableRow();
```

Create the first entry of the second row and append the entry to the row. The top border of this entry is determined by the bottom border of the row one, column one entry.

```
te21 = TableEntry("entry 2,1");  
append(tr2,te21);
```

Create the second entry of the second row, specify that the top border is red, and append the entry to the row.

```
te22 = TableEntry("entry 2,2");  
b = Border();  
b.TopColor = 'red';  
te22.Style = [te22.Style {b}];  
append(tr2,te22);
```

Create the third entry of the second row, specify that the top border is red, and append the entry to the row.

```
te23 = TableEntry("entry 2,3");  
b = Border();  
b.TopColor = 'red';  
te23.Style = [te23.Style {b}];  
append(tr2,te23);
```

Append the rows to the table.

```
append(t,tr1);  
append(t,tr2);
```

Add a title and the table to the slide.

```
replace(slide,"Title","Table Entry That Spans Multiple Columns");  
replace(slide,"Table",t);
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

The bottom border of the first row entry is red along the entire span.

Table Entry That Spans Multiple Columns

entry 1,1		
entry 2,1	entry 2,2	entry 2,3

12/27/2010 1

### Customize Right Border of Entry That Spans Multiple Rows

To customize the right border of an entry that spans multiple rows:

- Specify the right border of the entry that spans the rows. This specification determines the border that is shared with the entry in the next column that is in the first row of the span.
- In the next column, specify the left border of the entries that are in the remaining rows of the span.

This example defines a table that has two rows and two columns. The entry in row one and column one spans two rows. To make the right border of this entry red along the entire span, the example specifies that:

- The right border of the row one, column one entry is red.
- The left border of the row two, column two entry is red.

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myRowSpanBorder.pptx");
open(ppt);
```

Add a slide to the presentation.

```
slide = add(ppt, "Title and Table");
```

Create a table that has two columns.

```
t = Table(2);
t.Style = [t.Style {VAlign("middleCentered")}];
```

Create the first row of the table.

```
tr1 = TableRow();
```

Create the first entry of the first row and specify that the entry spans two rows. Specify that the right border of the entry is red. Append the entry to the row.

```
te11 = TableEntry("entry 1,1");  
te11.RowSpan = 2;  
border = Border();  
border.RightColor = 'red';  
te11.Style = [te11.Style {border}];  
append(tr1,te11);
```

Create the second entry of the first row and append the entry to the row. The left border of this entry is determined by the right border of the row one, column one entry.

```
te12 = TableEntry("entry 1,2");  
append(tr1,te12);
```

Create the second entry of the second row, specify that the left border is red, and append the entry to the row.

```
tr2 = TableRow();  
te22 = TableEntry("entry 2,2");  
b = Border();  
b.LeftColor = 'red';  
te22.Style = [te22.Style {b}];  
append(tr2,te22);
```

Append the rows to the table.

```
append(t,tr1);  
append(t,tr2);
```

Add a title and the table to the slide.

```
replace(slide,"Title","Table Entry That Spans Multiple Rows");  
replace(slide,"Table",t);
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

The right border of the first column entry is red along the entire span.



Table Entry That Spans Multiple Rows

entry 1,1	entry 1,2
	entry 2,2

12/27/2010 1

## See Also

entry | mlreportgen.ppt.Table | mlreportgen.ppt.TableRow

## Topics

“Create and Format Tables” on page 14-69

“Presentation Formatting Approaches” on page 14-18

“Presentation Format Inheritance” on page 14-21

## Introduced in R2015b

## mlreportgen.ppt.TablePlaceholder class

**Package:** mlreportgen.ppt

Placeholder for slide table

### Description

An object of the `mlreportgen.ppt.TablePlaceholder` class represents a table placeholder in a slide.

The PPT API creates a `TablePlaceholder` object when you add a slide to a presentation and the slide layout has a table placeholder. In the default PPT API, the `Title` and `Table` layout has a table placeholder.

To find a `TablePlaceholder` object, use the `find` method of the slide that contains the table placeholder. To replace the table placeholder with a table, use the `replace` method of the `TablePlaceholder` object. The `replace` method replaces the `TablePlaceholder` object with an `mlreportgen.ppt.Table` object.

You can use the properties of a `TablePlaceholder` object to specify the position or size of the replacement table. However, the text formatting properties are ignored. To format the table that you use to replace a `TablePlaceholder` object, use the properties of the `Table` object.

The `mlreportgen.ppt.TablePlaceholder` class is a handle class.

### Class Attributes

<code>HandleCompatible</code>	<code>true</code>
<code>ConstructOnLoad</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

### Properties

#### **Bold** — This property is ignored

`[]` (default) | `true` | `false`

This property is ignored.

#### **Font** — This property is ignored

`[]` (default) | character vector | string scalar

This property is ignored.

#### **ComplexScriptFont** — This property is ignored

`[]` (default) | `true` | `false`

This property is ignored.

#### **FontColor** — This property is ignored

`[]` (default) | character vector | string scalar

This property is ignored.

**FontSize — This property is ignored**

[] (default) | character vector | string scalar

This property is ignored.

**Italic — This property is ignored**

[] (default) | true | false

This property is ignored.

**Strike — This property is ignored**

[] (default) | 'single' | 'none' | 'double'

This property is ignored.

**Subscript — This property is ignored**

[] (default) | true | false

This property is ignored.

**Superscript — This property is ignored**

[] (default) | true | false

This property is ignored.

**Underline — This property is ignored**

[] (default) | 'single' | 'double' | ...

This property is ignored.

**BackgroundColor — This property is ignored**

[] (default) | character vector | string scalar

This property is ignored.

**Valign — This property is ignored**

[] (default) | 'top' | 'bottom' | ...

This property is ignored.

**Name — Table placeholder name**

character vector | string scalar

Table placeholder name, specified as a character vector or string scalar.

**X — Upper left x-coordinate of position of table placeholder**

character vector | string scalar

Upper left x-coordinate of the position of the table placeholder in the slide, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '5in' specifies 5 inches. Valid abbreviations are:

- px — pixels (default)

- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**Y — Upper left y-coordinate of position of table placeholder**

character vector | string scalar

Upper left y-coordinate of the position of the table placeholder in the slide, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '5in' specifies 5 inches. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**Width — Width of table placeholder**

character vector | string scalar

Width of the table placeholder, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '5in' specifies five inches. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**Height — Height of table placeholder**

character vector | string scalar

Height of the table placeholder, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '5in' specifies five inches. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**Style — This property is ignored**

cell array of PPT format objects

This property is ignored.

**Children — Children of this PPT API object**

cell array of PPT objects

Child elements of this object, specified as a cell array of PPT objects. This property is read-only.

**Parent — Parent of this PPT API object**

PPT object

Parent of this object, specified as a PPT object. This property is read-only.

**Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

**Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

**Methods****Public Methods**

replace Replace table placeholder with table

**Examples****Replace Table Placeholder with Table**

Add a Title and Table slide to a presentation and then replace the title and table placeholders with your own title and table.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation.

```
ppt = Presentation("myTablePlaceholderPresentation.pptx");
open(ppt);
```

Add a slide that has a Title and Table layout.

```
slide = add(ppt,"Title and Table");
```

Use the find method of the slide object to find the placeholder object that has the name Title.

```
titlePlaceholderObj = find(slide,"Title")
```

```
titlePlaceholderObj =  
  TextBoxPlaceholder with properties:  
  
      Bold: []  
      Font: []  
ComplexScriptFont: []  
      FontColor: []  
      FontSize: []  
      Italic: []  
      Strike: []  
      Subscript: []  
      Superscript: []  
      Underline: []  
Background-color: []  
      VAlign: []  
      Name: 'Title'  
      X: []  
      Y: []  
      Width: []  
      Height: []  
      Style: []  
Children: []  
Parent: [1x1 mlreportgen.ppt.Slide]  
Tag: 'ppt.TextBoxPlaceholder:30:96'  
Id: '30:96'
```

The find method returns an mlreportgen.ppt.TextBoxPlaceholder object.

Replace the placeholder content with the title text.

```
replace(titlePlaceholderObj,"Fourth-Order Magic Square");
```

Use the find method of the slide object to find the placeholder object that has the name Table.

```
tablePlaceholderObj = find(slide,"Table")
```

```
tablePlaceholderObj =  
  TablePlaceholder with properties:  
  
      Bold: []  
      Font: []  
ComplexScriptFont: []  
      FontColor: []  
      FontSize: []  
      Italic: []  
      Strike: []  
      Subscript: []  
      Superscript: []  
      Underline: []  
Background-color: []  
      VAlign: []  
      Name: 'Table'
```

```

        X: []
        Y: []
        Width: []
        Height: []
        Style: []
        Children: []
        Parent: [1x1 mlreportgen.ppt.Slide]
        Tag: 'ppt.TablePlaceholder:31:97'
        Id: '31:97'

```

The `find` method returns an `mlreportgen.ppt.TablePlaceholder` object.

Replace the table placeholder with a table for a fourth-order magic square.

```
replace(tablePlaceholderObj,Table(magic(4)));
```

Close and view the presentation

```
close(ppt);
rptview(ppt);
```

Here is the slide in the generated presentation:

16	2	3	13
5	11	10	8
9	7	6	12
4	14	15	1

## Tips

- When you replace a table placeholder with a table in a presentation and then use the presentation as a template for a new presentation, the PPT API creates an `mlreportgen.ppt.TemplateTable` object for the table in the new presentation. See “Add or Replace a Table” on page 14-61.
- To see the placeholder objects that the PPT API creates for a slide object, view the `Children` property of the slide. For example, when you add a `Title` and `Table` slide to a presentation, the `Children` property is an array that contains an `mlreportgen.ppt.TextBoxPlaceholder` object and an `mlreportgen.ppt.TablePlaceholder` object.

```
ppt = mlreportgen.ppt.Presentation("test.pptx");
open(ppt);
slide = add(ppt,"Title and Table");
slide.Children(1)
```

```
ans =
```

```
TextBoxPlaceholder with properties:
```

```
        Bold: []
        Font: []
ComplexScriptFont: []
        FontColor: []
        FontSize: []
        Italic: []
        Strike: []
        Subscript: []
        Superscript: []
        Underline: []
        BackgroundColor: []
        VAlign: []
        Name: 'Title'
        X: []
        Y: []
        Width: []
        Height: []
        Style: []
        Children: []
        Parent: [1x1 mlreportgen.ppt.Slide]
        Tag: 'ppt.TextBoxPlaceholder:6:7'
        Id: '6:7'
```

```
slide.Children(2)
```

```
ans =
```

```
TablePlaceholder with properties:
```

```
        Bold: []
        Font: []
ComplexScriptFont: []
        FontColor: []
        FontSize: []
        Italic: []
        Strike: []
        Subscript: []
        Superscript: []
        Underline: []
        BackgroundColor: []
        VAlign: []
        Name: 'Table'
        X: []
        Y: []
        Width: []
        Height: []
        Style: []
        Children: []
        Parent: [1x1 mlreportgen.ppt.Slide]
```



Tag: 'ppt.TablePlaceholder:7:8'  
Id: '7:8'

### **See Also**

mreportgen.ppt.PicturePlaceholder | mreportgen.ppt.ContentPlaceholder |  
mreportgen.ppt.Table | mreportgen.ppt.TextBoxPlaceholder |  
mreportgen.ppt.TemplateTable | getLayoutNames

### **Topics**

“Access PowerPoint Template Elements” on page 14-32

“Add or Replace a Table” on page 14-61

### **Introduced in R2015b**

## mlreportgen.ppt.TableRow class

**Package:** mlreportgen.ppt mlreportgen.ppt

Table row

### Description

Use an object of the `mlreportgen.ppt.TableRow` class to include a row in a table in a PPT API presentation.

To add content to a table row, append `mlreportgen.ppt.TableEntry` objects to the row.

The `mlreportgen.ppt.TableRow` class is a `handle` class.

### Class Attributes

<code>HandleCompatible</code>	<code>true</code>
<code>ConstructOnLoad</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`tableRowObj = mlreportgen.ppt.TableRow()` creates an empty `TableRow` object.

## Properties

### Height — Row height

`[]` (default) | character vector | string scalar

Row height, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, `'2in'` specifies 2 inches. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

If the table height is specified and the row height is not specified for any row, the height of all the rows is the same. The row height is determined by dividing the table height by the number of rows. If the height is specified for at least one row, the PPT API ignores the table height. Microsoft PowerPoint determines the height of the rows for which the height is not specified.

**Font — Font family for row text**

[] (default) | character vector | string scalar

Font family for row text, specified as a character vector or string scalar. Specify a font that appears in the font list in Microsoft PowerPoint. To see the font list, in PowerPoint, on the **Home** tab, in the **Font** group, click the arrow to the right of the font.

**ComplexScriptFont — Font family for complex scripts**

[] (default) | character vector | string scalar

Font family for complex scripts, specified as a character vector or string scalar. Specify a font family to use when substituting in a locale that requires a complex script, such as Arabic or Asian, for rendering text.

**FontColor — Font color of row text**

[] (default) | character vector | string scalar

Font color of row text, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

**FontSize — Font size of row text**

[] (default) | character vector | string scalar

Font size of row text, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '11pt' specifies 11 points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**BackgroundColor — Background color of row**

[] (default) | character vector | string scalar

Background color of the row, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

**Style — Table row formatting**

cell array of PPT format objects

Table row formatting, specified as a cell array of PPT format objects. Formats that do not apply to a `TableRow` object are ignored.

**Children — Children of this PPT API object**

cell array of PPT objects

Child elements of this object, specified as a cell array of PPT objects. This property is read-only.

**Parent — Parent of this PPT API object**

PPT object

Parent of this object, specified as a PPT object. This property is read-only.

**Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

**Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

**Methods****Public Methods**

append	Append table entry to table row
clone	<p><code>objCopy = clone(obj)</code> returns a copy of the <code>mlreportgen.ppt.TableRow</code> object specified by <code>obj</code>.</p> <p>Use the <code>clone</code> method of a <code>TableRow</code> object the same way that you use the <code>clone</code> method of a <code>Paragraph</code> object.</p>

**Examples****Create a Table Using Entries and Rows**

Create a table for a presentation by adding entries to rows and the rows to a table.

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation('myTableRowPresentation.pptx');
open(ppt);
```

Add a slide to the presentation.

```
add(ppt, 'Title and Content');
```

Create a table.

```
table = Table();
```

Create the first table row.

```
tr1 = TableRow();  
tr1.Style = [tr1.Style {Bold(true)}];
```

Create the table entries for the first row.

```
te1tr1 = TableEntry();  
p = Paragraph('first entry');  
p.FontColor = 'red';  
append(te1tr1,p);  
append(tr1,te1tr1);
```

```
te2tr1 = TableEntry();  
append(te2tr1,'second entry');  
append(tr1,te2tr1);
```

```
te3tr1 = TableEntry();  
te3tr1.FontColor = 'green';  
append(te3tr1,'third entry');  
append(tr1,te3tr1);
```

Append the first table row to the table.

```
append(table,tr1);
```

Create the second table row, append the table entries to the row, and append the row to the table.

```
tr2 = TableRow();  
te1tr2 = TableEntry();  
te1tr2.FontColor = 'red';  
p = Paragraph('first entry');  
append(te1tr2,p);  
append(tr2,te1tr2);
```

```
te2tr2 = TableEntry();  
append(te2tr2,'second entry');  
append(tr2,te2tr2);
```

```
te3tr2 = TableEntry();  
te3tr2.FontColor = 'green';  
append(te3tr2,'third entry');  
append(tr2,te3tr2);  
append(table,tr2);
```

Add the table to the presentation.

```
contents = find(ppt,'Content');  
replace(contents(1),table);
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

Here is the table in the generated presentation:

first entry	second entry	third entry
first entry	second entry	third entry

12/1/2019 1

### See Also

`row` | `mlreportgen.ppt.Table` | `mlreportgen.ppt.TableEntry`

### Topics

“Create and Format Tables” on page 14-69

**Introduced in R2015b**

# mlreportgen.ppt.TableStyleOptions class

**Package:** mlreportgen.ppt

Stripe table rows and columns

## Description

Specifies whether to format table rows and columns. Before you use `TableStyleOptions`, specify the table style using the `StyleName` property on the `Table` object you want to apply the options to. The table style determines the formatting of the table, for example, the color of the banding and first- and last-column emphasis.

## Construction

`tableStyles = TableStyleOptions()` creates a `TableStyleOptions` object. This object uses the properties of the table style assigned to the `StyleName` property of the table you assign the properties to. The table uses the `TableStyleOption` object only if the associated table sets the `StyleName` property.

## Output Arguments

### **tableStyles — Table style property options**

mlreportgen.ppt.TableStyleOptions object

Table style property options, returned as an `mlreportgen.ppt.TableStyleOptions` object.

## Properties

### **BandedColumns — Alternating color for columns**

true | false

Alternating color (banding) for columns, specified as:

- `true` — Colors alternate for columns.
- `false` — Columns use same color.

### **BandedRows — Alternating color (banding) for rows**

true | false

Alternating color for rows, specified as:

- `true` — Colors alternate for rows.
- `false` — Rows use same color.

### **FirstColumn — Emphasis for first column in table**

true | false

Emphasis for first column in table, specified as:

- `true` — First column uses emphasis styling, e.g., stronger color, emphasized font.
- `false` — Regular styling on first column.

**FirstRow — Emphasis for first row of table**`true | false`

Emphasis for first row of table, specified as:

- `true` — First row uses emphasis styling, e.g., stronger color, emphasized font.
- `false` — Regular styling on first row.

**Id — ID for this PPT API object**`character vector | string scalar`

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

**LastColumn — Emphasis for last column in table**`true | false`

Emphasis for last column in table, specified as:

- `true` — Last column uses emphasis styling, e.g., stronger color, emphasized font.
- `false` — Regular styling on last column.

**LastRow — Emphasis for last row of table**`true | false`

Emphasis for last row of table, specified as:

- `true` — Last row uses emphasis styling, e.g., stronger color, emphasized font.
- `false` — Regular styling on last row.

**Tag — Tag for this PPT API object**`character vector | string scalar`

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## Examples

**Style a Table**

This example shows how to control the formatting of a table based on the `Medium Style 2 - Accent 1` table style in the default PPT template. Change the values of each of the styling properties from `false` to `true`, or the reverse, to understand their effects.

```
import mlreportgen.ppt.*
```



```
ppt = Presentation('myppt.pptx');
open(ppt);
add(ppt, 'Title and Content');

% Create the table and specify the table style name
table = Table(magic(5));
table.StyleName = 'Medium Style 2 - Accent 1';

% Create the format and set the properties
tblStyle = TableStyleOptions();
tblStyle.FirstRow = false;
tblStyle.LastRow = false;
tblStyle.FirstColumn = true;
tblStyle.LastColumn = false;
tblStyle.BandedRows = true;
tblStyle.BandedColumns = false;

% Apply the formatting to the table
table.Style = {tblStyle};

% Add the table to the slide
% Generate and display the presentation
replace(ppt, 'Content', table);
close(ppt);
rptview(ppt);
```

## See Also

mlreportgen.ppt.Table

## Topics

“Create and Format Tables” on page 14-69

## Introduced in R2016a

## mlreportgen.ppt.TemplatePicture class

**Package:** mlreportgen.ppt

Picture from template presentation slide

### Description

Objects of the `mlreportgen.ppt.TemplatePicture` represent template pictures. A template picture is a picture that comes from a slide in the template presentation. Customize a template picture by modifying properties of the associated `mlreportgen.ppt.TemplatePicture` object. Replace a template picture with another picture by using the `replace` method of the `mlreportgen.ppt.TemplatePicture` object.

The `mlreportgen.ppt.TemplatePicture` class is a handle class.

### Class Attributes

<code>HandleCompatible</code>	<code>true</code>
<code>ConstructOnLoad</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

### Creation

You do not create an `mlreportgen.ppt.TemplatePicture` object explicitly. When you create a presentation from an existing presentation, the MATLAB API for PowerPoint (PPT API) creates an `mlreportgen.ppt.TemplatePicture` object for each picture that comes from a slide in the template presentation.

You can access the `mlreportgen.ppt.TemplatePicture` object for a template picture by using:

- The `find` method of the `mlreportgen.ppt.Presentation` object.
- The `find` method of the `mlreportgen.ppt.Slide` object that corresponds to the slide that contains the template picture.
- The `Children` property of the `mlreportgen.ppt.Slide` object that corresponds to the slide that contains the template picture.

---

**Note** Pictures that you add to a new presentation are represented as `mlreportgen.ppt.Picture` objects.

---

### Properties

#### XMLMarkup — XML markup of template picture

character vector

XML markup of template picture, specified as a character vector. You can modify the template picture by changing the XML markup. Modify the XML markup for customizations for which there is no

property. The updated markup is written to the generated presentation. If you update other properties, such as `X`, `Y`, `Width`, or `Height`, the corresponding attributes in the XML markup are updated before the markup is written to the generated presentation.

**Name — Picture name**

character vector | string scalar

Picture name, specified as a character vector or string scalar.

**X — Upper left x-coordinate of position of picture**

character vector | string scalar

Upper left x-coordinate of the position of the picture, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '5in' specifies 5 inches. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

When the PPT API creates an `mlreportgen.ppt.TemplatePicture` object, this property value is specified in English Metric Units (EMU). If you set this property, you must use one of the units in the previous list.

**Y — Upper left y-coordinate of position of picture**

character vector | string scalar

Upper left y-coordinate of the position of the picture, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '5in' specifies 5 inches. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

When the PPT API creates an `mlreportgen.ppt.TemplatePicture` object, this property value is specified in English Metric Units (EMU). If you set this property, you must use one of the units in the previous list.

**Width — Width of picture**

character vector | string scalar

Width of picture, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '5in' specifies five inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

When the PPT API creates an `mlreportgen.ppt.TemplatePicture` object, this property value is specified in English Metric Units (EMU). If you set this property, you must use one of the units in the previous list.

**Height — Height of picture**

character vector | string scalar

Height of picture, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '5in' specifies five inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

When the PPT API creates an `mlreportgen.ppt.TemplatePicture` object, this property value is specified in English Metric Units (EMU). If you set this property, you must use one of the units in the previous list.

**Style — Template picture formatting**

[] (default)

This property is ignored.

**Children — Children of this PPT API object**

[]

This read-only property is empty.

**Parent — Parent of this PPT API object**

PPT object

Parent of this object, specified as a PPT object. This property is read-only.

**Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

**Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## Methods

### Public Methods

replace Replace template picture with another picture

## Examples

### Modify and Replace Pictures from a Template Presentation

Generate a presentation, MyPicturePresentation, that you then use as the template presentation for another presentation. MyPicturePresentation has two slides, and each slide has one picture.

```
import mlreportgen.ppt.*
ppt = Presentation("MyPicturePresentation");
open(ppt);

slide1 = add(ppt,"Title and Picture");
replace(slide1,"Title","Plane");
replace(slide1,"Picture",Picture("b747.jpg"));

slide2 = add(ppt,"Title and Picture");
replace(slide2,"Title","Street");
replace(slide2,"Picture",Picture("street1.jpg"));
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Create a presentation, MyNewPicturePresentation, from MyPicturePresentation. MyPicturePresentation is the template presentation for MyNewPicturePresentation.

```
ppt = Presentation("MyNewPicturePresentation","MyPicturePresentation");
open(ppt);
```

Find the template pictures by using the find method of the slide objects. Because the pictures come from the template presentation, find returns the pictures as mlreportgen.ppt.TemplatePicture objects.

```
slide1 = ppt.Children(1);
slide2 = ppt.Children(2);
templatePictureObj1 = find(slide1,"Picture")
```

```
templatePictureObj1 =
    TemplatePicture with properties:
```

```
XMLMarkup: '<p:pic><p:nvPicPr><p:cNvPr id="8" name="Picture"/><p:cNvPicPr><a:picLocks noChan
Name: 'Picture'
```

```
        X: []
        Y: []
    Width: []
    Height: []
    Style: []
    Children: []
    Parent: [1x1 mlreportgen.ppt.Slide]
    Tag: 'ppt.TemplatePicture:435:246'
    Id: '435:246'
```

```
templatePictureObj2 = find(slide2,"Picture")
```

```
templatePictureObj2 =
    TemplatePicture with properties:
```

```
    XMLMarkup: '<p:pic><p:nvPicPr><p:cNvPr id="8" name="Picture"/><p:cNvPicPr><a:picLocks noChang
    Name: 'Picture'
        X: []
        Y: []
    Width: []
    Height: []
    Style: []
    Children: []
    Parent: [1x1 mlreportgen.ppt.Slide]
    Tag: 'ppt.TemplatePicture:439:248'
    Id: '439:248'
```

Change the size of the picture on the first slide.

```
templatePictureObj1.Width = "4in";
templatePictureObj1.Height = "3in";
```

Replace the picture on the second slide with a picture of a different street.

```
replace(templatePictureObj2,Picture("street2.jpg"));
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

## See Also

[mlreportgen.ppt.Presentation](#) | [mlreportgen.ppt.Slide](#) | [mlreportgen.ppt.Picture](#) | [mlreportgen.ppt.PicturePlaceholder](#)

## Topics

“Create a Presentation Object to Hold Content” on page 14-12

“Access PowerPoint Template Elements” on page 14-32

“Add and Replace Presentation Content” on page 14-58

**Introduced in R2019b**

# mlreportgen.ppt.TemplateTable class

**Package:** mlreportgen.ppt

Table from template presentation slide

## Description

Objects of the `mlreportgen.ppt.TemplateTable` class represent template tables. A template table is a table that comes from a slide in the template presentation. Customize a template table by modifying properties of the associated `mlreportgen.ppt.TemplateTable` object. Replace a template table with another table by using the `replace` method of the `mlreportgen.ppt.TemplateTable` object.

The `mlreportgen.ppt.TemplateTable` class is a handle class.

## Class Attributes

<code>HandleCompatible</code>	<code>true</code>
<code>ConstructOnLoad</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Creation

You do not create an `mlreportgen.ppt.TemplateTable` object explicitly. When you create a presentation from an existing presentation, the MATLAB API for PowerPoint (PPT API) creates an `mlreportgen.ppt.TemplateTable` object for each table that comes from a slide in the template presentation.

You can access the `mlreportgen.ppt.TemplateTable` object for a template table by using:

- The `find` method of the `mlreportgen.ppt.Presentation` object.
- The `find` method of the `mlreportgen.ppt.Slide` object that corresponds to the slide that contains the template table.
- The `Children` property of the `mlreportgen.ppt.Slide` object that corresponds to the slide that contains the template table.

---

**Note** Tables that you add to a new presentation are represented as `mlreportgen.ppt.Table` objects.

---

## Properties

### XMLMarkup — XML markup of template table

character vector

XML markup of template table, specified as a character vector. You can modify the template table by changing the XML markup. Modify the XML markup for customizations for which there is no

property. The updated markup is written to the generated presentation. If you update other properties, such as `X`, `Y`, `Width`, or `Height`, the corresponding attributes in the XML markup are updated before the markup is written to the generated presentation.

**Name — Table name**

character vector | string scalar

Table name, specified as a character vector or string scalar.

**X — Upper left x-coordinate of position of table**

[] (default) | character vector | string scalar

Upper left x-coordinate of the position of the table, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '5in' specifies 5 inches. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

When the PPT API creates an `mlreportgen.ppt.TemplateTable` object, this property value is specified in English Metric Units (EMU). If you set this property, you must use one of the units in the previous list.

**Y — Upper left y-coordinate of position of table**

[] (default) | character vector | string scalar

Upper left y-coordinate of the position of the table, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '5in' specifies 5 inches. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

When the PPT API creates an `mlreportgen.ppt.TemplateTable` object, this property value is specified in English Metric Units (EMU). If you set this property, you must use one of the units in the previous list.

**Width — Width of table**

[] (default) | character vector | string scalar

Width of table, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '5in' specifies five inches. Valid abbreviations are:



- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

When the PPT API creates an `mlreportgen.ppt.TemplateTable` object, this property value is specified in English Metric Units (EMU). If you set this property, you must use one of the units in the previous list.

### **Height — Height of table**

[ ] (default) | character vector | string scalar

Height of table, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '5in' specifies five inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

When the PPT API creates an `mlreportgen.ppt.TemplateTable` object, this property value is specified in English Metric Units (EMU). If you set this property, you must use one of the units in the previous list.

### **Style — Template table formatting**

[ ] (default)

This property is ignored.

### **Children — Children of this PPT API object**

[ ]

This read-only property is empty.

### **Parent — Parent of this PPT API object**

PPT object

Parent of this object, specified as a PPT object. This property is read-only.

### **Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### **Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## Methods

### Public Methods

replace Replace template table with another table

## Examples

### Modify and Replace Tables from a Template Presentation

Generate a presentation, MyTablePresentation, that you then use as the template presentation for another presentation. MyTablePresentation has two slides, and each slide has one table.

```
import mlreportgen.ppt.*
ppt = Presentation("MyTablePresentation");
open(ppt);

slide1 = add(ppt,"Title and Table");
replace(slide1,"Title","Magic Square Slide 1");
replace(slide1,"Table",Table(magic(3)));

slide2 = add(ppt,"Title and Table");
replace(slide2,"Title","Magic Square Slide 2");
replace(slide2,"Table",Table(magic(5)));
```

Close and view MyTablePresentation.

```
close(ppt);
rptview(ppt);
```

Create a presentation, MyNewTablePresentation, from MyTablePresentation. MyTablePresentation is the template presentation for MyNewTablePresentation.

```
ppt = Presentation("MyNewTablePresentation", "MyTablePresentation");
open(ppt);
```

Find the template tables by using the find method of the slide objects. Because the tables come from the template presentation, find returns the tables as mlreportgen.ppt.TemplateTable objects.

```
slide1 = ppt.Children(1);
slide2 = ppt.Children(2);
templateTableObj1 = find(slide1,"Table")
```

```
templateTableObj1 =
    TemplateTable with properties:
```

```
XMLMarkup: '<p:graphicFrame><p:nvGraphicFramePr><p:cNvPr id="3" name="Table"/><p:cNvGraphicF
Name: 'Table'
X: '838200emu'
```

```

        Y: '1825625emu'
        Width: '10515600emu'
        Height: '4351338emu'
        Style: []
        Children: []
        Parent: [1x1 mlreportgen.ppt.Slide]
        Tag: 'ppt.TemplateTable:1234:488'
        Id: '1234:488'

```

```
templateTableObj2 = find(slide2, "Table")
```

```
templateTableObj2 =
    TemplateTable with properties:
```

```

    XMLMarkup: '<p:graphicFrame><p:nvGraphicFramePr><p:cNvPr id="3" name="Table"/><p:cNvGraphicF
        Name: 'Table'
        X: '838200emu'
        Y: '1825625emu'
        Width: '10515600emu'
        Height: '4351338emu'
        Style: []
        Children: []
        Parent: [1x1 mlreportgen.ppt.Slide]
        Tag: 'ppt.TemplateTable:1238:490'
        Id: '1238:490'

```

Change the position of the table on the first slide.

```
templateTableObj1.X = "1in";
templateTableObj1.Y = "3in";
```

Replace the table on the second slide with a table for a 4-by-4 magic square.

```
replace(templateTableObj2, Table(magic(4)));
```

Close and view MyNewTablePresentation.

```
close(ppt);
rptview(ppt);
```

## See Also

[mlreportgen.ppt.Presentation](#) | [mlreportgen.ppt.Slide](#) | [mlreportgen.ppt.Table](#) | [mlreportgen.ppt.TablePlaceholder](#)

## Topics

“Create a Presentation Object to Hold Content” on page 14-12

“Access PowerPoint Template Elements” on page 14-32

“Add and Replace Presentation Content” on page 14-58

## Introduced in R2019b

## mlreportgen.ppt.Text class

**Package:** mlreportgen.ppt mlreportgen.ppt

Text to include in a presentation

### Description

Use an object of the mlreportgen.ppt.Text class to include text in a PPT API presentation.

The mlreportgen.ppt.Text class is a handle class.

### Class Attributes

HandleCompatible	true
ConstructOnLoad	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

textObj = mlreportgen.ppt.Text() creates an empty mlreportgen.ppt.Text object.

textObj = mlreportgen.ppt.Text(text) creates an mlreportgen.ppt.Text object with the Content property set to text.

## Properties

### Content — Text content

'' (default) | character vector | string scalar

Text content, specified as a character vector or string scalar.

### Bold — Whether to use bold for text

[] (default) | true | false

Whether to use bold for the text, specified as true or false. Specify true for bold text.

Data Types: logical

### Font — Font family for text

[] (default) | character vector | string scalar

Font family for the text, specified as a character vector or string scalar. Specify a font that appears in the font list in Microsoft PowerPoint. To see the font list, in PowerPoint, on the **Home** tab, in the **Font** group, click the arrow to the right of the font.

### ComplexScriptFont — Font family for complex scripts

[] (default) | character vector | string scalar

Font family for complex scripts, specified as a character vector or string scalar. Specify a font family to use when substituting in a locale that requires a complex script, such as Arabic or Asian, to render text.

### **FontColor — Font color for text**

[] (default) | character vector | string scalar

Font color for the text, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

### **FontSize — Font size of text**

[] (default) | character vector | string scalar

Font size of the text, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '11pt' specifies 11 points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

### **Italic — Whether to use italic for text**

[] (default) | true | false

Whether to use italic for the text, specified as true or false. Specify true for italic text.

Data Types: logical

### **Strike — Strikethrough style for text**

[] (default) | 'single' | 'none' | 'double'

Strikethrough style for the text, specified as one of these values:

- 'single' — Single horizontal line
- 'none' — No strikethrough line
- 'double' — Double horizontal line

### **Subscript — Whether to display text as a subscript**

[] (default) | true | false

Whether to display the text as a subscript, specified as true or false. A setting of true renders the text as a subscript.

Data Types: logical

### **Superscript — Whether to display text as a superscript**

[] (default) | true | false

Whether to display the text as a superscript, specified as `true` or `false`. A setting of `true` renders the text as a superscript.

Data Types: `logical`

### **Underline — Underline style for text**

`[]` (default) | `'single'` | `'double'` | ...

Underline style for the text, specified as one of these values:

Value	Description
<code>'single'</code>	Single underline
<code>'double'</code>	Double underline
<code>'heavy'</code>	Thick underline
<code>'words'</code>	Only words underlined (not spaces)
<code>'dotted'</code>	Dotted underline
<code>'dottedheavy'</code>	Thick, dotted underline
<code>'dash'</code>	Dashed underline
<code>'dashheavy'</code>	Thick, dashed underline
<code>'dashlong'</code>	Long, dashed underline
<code>'dashlongheavy'</code>	Thick, long, dashed underline
<code>'dotdash'</code>	Dot-dash underline
<code>'dotdashheavy'</code>	Thick, dot-dash underline
<code>'dotdotdash'</code>	Dot-dot-dash underline
<code>'dotdotdashheavy'</code>	Thick, dot-dot-dash underline
<code>'wavy'</code>	Wavy underline
<code>'wavyheavy'</code>	Thick, wavy underline
<code>'wavydouble'</code>	Wavy, double underline
<code>'none'</code>	No underline

### **Style — Text formatting**

cell array of PPT format objects

Text formatting, specified as a cell array of PPT format objects.

Add format objects by concatenating the existing value of the `Style` property with a cell array that contains the format objects that you are adding. For example:

```
txt = mlreportgen.ppt.Text('my text');
txt.Style = [txt.Style {Bold(true),FontColor('red')}];
```

See “Presentation Formatting Approaches” on page 14-18.

### **Children — Children of this PPT API object (not used)**

`[]`

This property is not used.

**Parent — Parent of this PPT API object**

PPT object

Parent of this object, specified as a PPT object. This property is read-only.

**Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

**Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

**Methods****Public Methods**

clone	<p><code>objCopy = clone(obj)</code> returns a copy of the <code>mlreportgen.ppt.Text</code> object specified by <code>obj</code>.</p> <p>Use the <code>clone</code> method of a <code>Text</code> object the same way that you use the <code>clone</code> method of a <code>Paragraph</code> object.</p>
-------	---

**Examples****Add Text to a Presentation**

Use an `mlreportgen.ppt.Text` object to add text to a presentation. This example uses `mlreportgen.ppt.Text` objects to add text to the title of one slide and the content of another slide.

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myTextPresentation.pptx");
open(ppt);
```

Add two slides to the presentation.

```
slide1 = add(ppt,"Title Slide");
slide2 = add(ppt,"Title and Content");
```

Create an `mlreportgen.ppt.Paragraph` object to use for the title of the presentation.

```
p1 = Paragraph;
```

Create an `mlreportgen.ppt.Text` object with bold and red content, and append it to the paragraph.

```
titleText = Text("My Presentation Title");  
titleText.Bold = true;  
titleText.FontColor = "red";  
append(p1,titleText);
```

Replace the title in the first slide with the paragraph.

```
replace(slide1,"Title",p1);
```

Create a paragraph for the content of the second slide.

```
p2 = Paragraph("My content");  
append(p2,Text(" for the second slide"));
```

Replace the content in the second slide with the paragraph.

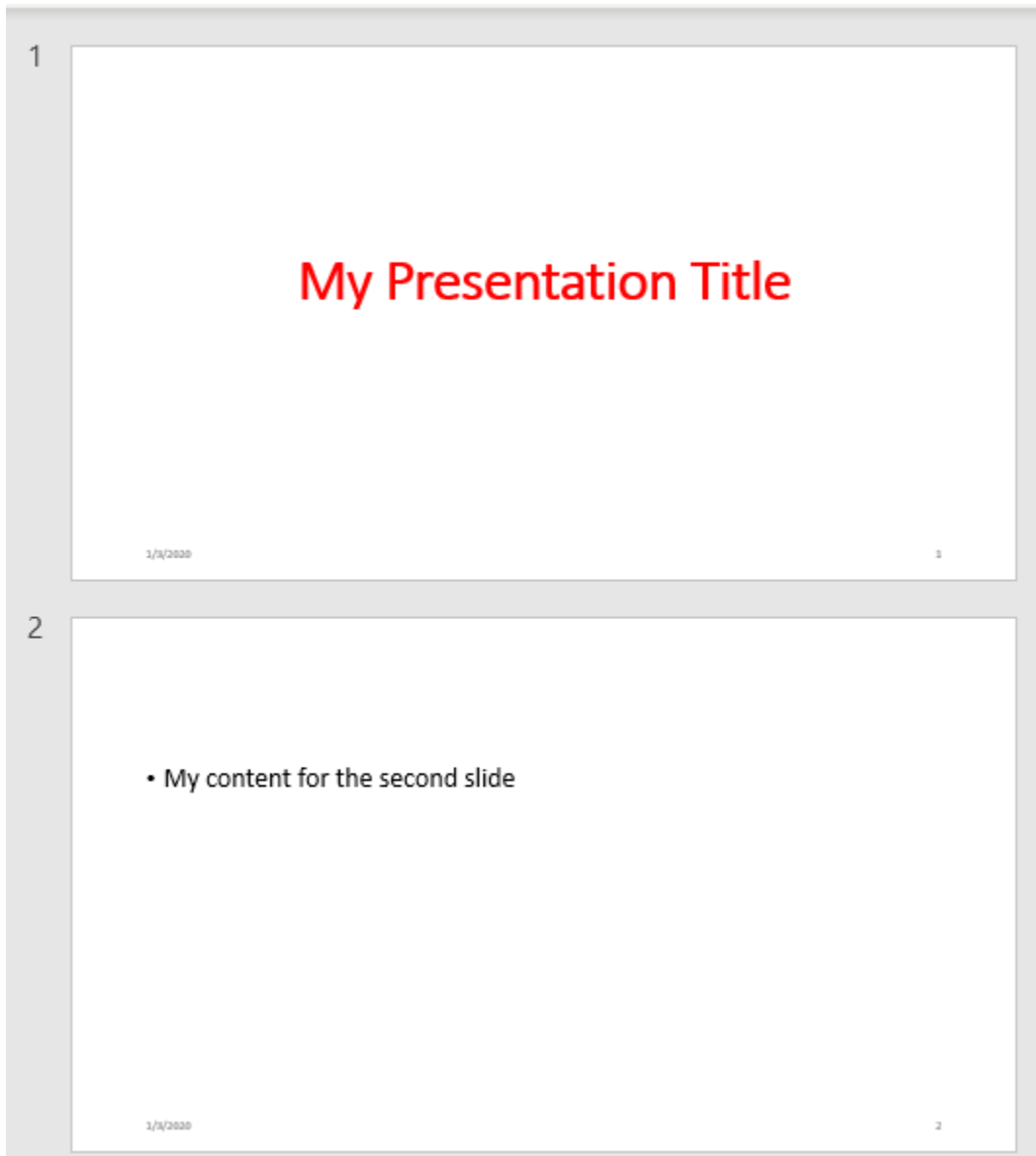
```
replace(slide2,"Content",p2);
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

Here are the slides in the generated presentation:





## See Also

[mlreportgen.ppt.Paragraph](#) | [mlreportgen.ppt.TextBox](#) | [mlreportgen.ppt.TextBoxPlaceholder](#) | [mlreportgen.ppt.InternalLink](#)

## Topics

“Create and Format Text” on page 14-65  
“Presentation Formatting Approaches” on page 14-18  
“Presentation Format Inheritance” on page 14-21

## Introduced in R2015b

## mlreportgen.ppt.TextBox class

**Package:** mlreportgen.ppt mlreportgen.ppt

Text box

### Description

Text box to include in a presentation.

The mlreportgen.ppt.TextBox class is a handle class.

### Creation

#### Description

textBoxObj = TextBox() creates an empty text box object.

### Properties

#### **Bold** — Option to use bold for text

logical value

Option to use bold for text, specified as true or false. To make text bold, set this property to true

#### **FontColor** — Font color for presentation element

character vector

Font color, specified as a character vector. Use either a CSS color name or a hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for each for the red, green, and blue values. For example, '#0000ff' specifies blue.

#### **Italic** — Option to use italics for text

[] (default) | logical value

Option to use italics for text, specified as a logical. Set this property to true. Setting the Italic property adds a corresponding mlreportgen.ppt.Italic format object to the Style property of this presentation element. Removing the Italic property setting removes the object.

Data Types: logical

#### **Underline** — Type of underline for text

[] (default) | character vector

Type of underlining for text, specified as a character vector. Setting the Underline property adds a corresponding mlreportgen.ppt.Underline format object to the Style property for this element. Removing the Underline property setting removes the object. You can specify one of these types of underlines.

<b>Value</b>	<b>Description</b>
'single'	Single underline
'double'	Double underline
'heavy'	Thick underline
'words'	Words only underlined (not spaces)
'dotted'	Dotted underline
'dottedheavy'	Thick, dotted underline
'dash'	Dashed underline
'dashheavy'	Thick, dashed underline
'dashlong'	Long, dashed underline
'dashlongheavy'	Thick, long, dashed underline
'dotdash'	Dot dash underline
'dotdotdash'	Dot dot dash underline
'dotdotdashheavy'	Thick dot dot dash underline
'dotdashdotheavy'	Thick dash dot underline
'wavy'	Wavy underline
'wavyheavy'	Thick wavy underline
'wavydouble'	Two wavy underlines

**Name — Text box name**

character vector

Text box name, specified as a character vector.

**X — Upper-left x-coordinate position of text box**

character vector

Upper-left x-coordinate of text box, specified in the form `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Y — Upper-left y-coordinate position of text box**

character vector

Upper-left y-coordinate position of text box, specified in the form `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- px — pixels (default)

- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**Width — Width of text box**

character vector

Width of text box, specified in the form `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**Height — Height of text box**

character vector

Height of text box, specified in the form `valueUnits` where `Units` is an abbreviation for the units. Valid abbreviations are:

- `px` — pixels (default)
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points

**Style — Text formatting**

cell array of PPT format objects

Text formatting, specified as a cell array of PPT format objects. You can specify these `mlreportgen.ppt` format objects:

- `BackgroundColor` object
- `FontFamily` object
- `FontSize` object
- `Bold` object
- `FontColor` object
- `Italic` object
- `Underline` object

**Children — Children of this PPT API object**

cell array of PPT objects

Child elements of this object, specified as a cell array of PPT objects. This property is read-only.

#### Parent — Parent of this PPT API object

PPT object

Parent of this object, specified as a PPT object. This property is read-only.

#### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

#### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

## Methods

### Public Methods

Method	Purpose
<code>add</code>	Add paragraph to text box
<code>replace</code>	Replace text box paragraphs
<code>clone</code>	<p><code>objCopy = clone(obj)</code> returns a copy of the <code>mlreportgen.ppt.TextBox</code> object specified by <code>obj</code>.</p> <p>Use the <code>clone</code> method of a <code>TextBox</code> object the same way that you use the <code>clone</code> method of a <code>Paragraph</code> object.</p>

## Examples

### Add a Text Box

Create a presentation with two slides.

```
import mlreportgen.ppt.*
ppt = Presentation('myTextBoxPresentation.pptx');
```

Add a blank slide.

```
blank = add(ppt, 'Blank');
```

Create a text box and define its location and size.

```
tb = TextBox();  
tb.X = '1in';  
tb.Y = '1in';  
tb.Width = '8in';  
tb.Height = '0.5in';
```

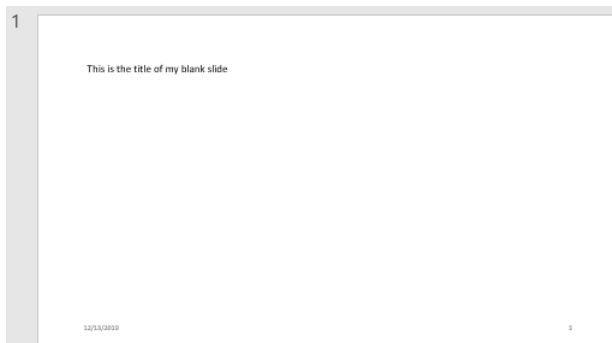
Add text to the text box and append the text box to the blank slide.

```
add(tb, 'This is the title of my blank slide');  
add(blank, tb);
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

Here is the generated presentation:



### See Also

[mlreportgen.ppt.Paragraph](#) | [mlreportgen.ppt.Text](#) | [mlreportgen.ppt.TextBoxPlaceholder](#)

### Topics

“Create and Format Text” on page 14-65

**Introduced in R2014b**

# mlreportgen.ppt.TextBoxPlaceholder class

**Package:** mlreportgen.ppt

Placeholder for slide title

## Description

An object of the `mlreportgen.ppt.TextBoxPlaceholder` class is a placeholder for a slide title. When you add a slide with a title to a presentation, the PPT API creates a `TextBoxPlaceholder` object for the title. All slide layouts in the default PPT API, except `Blank`, have a title. To find a `TextBoxPlaceholder` object, use the `find` method of the slide that contains the title. Then, you can set the properties of the `TextBoxPlaceholder` object to specify the default format for the title text that replaces the placeholder content. To replace the placeholder content with your title text, use the `replace` method of the `TextBoxPlaceholder` object. The `replace` method does not replace the `TextBoxPlaceholder` object. It adds the text as one or more paragraphs to the `Children` property of the `TextBoxPlaceholder` object.

The `mlreportgen.ppt.TextBoxPlaceholder` class is a handle class.

## Class Attributes

<code>HandleCompatible</code>	<code>true</code>
<code>ConstructOnLoad</code>	<code>true</code>

For information on class attributes, see “Class Attributes”.

## Properties

### **Bold** — Whether to use bold for text

`[]` (default) | `true` | `false`

Whether to use bold for the text, specified as `true` or `false`. Specify `true` for bold text.

Data Types: `logical`

### **Font** — Font family for text

`[]` (default) | character vector | string scalar

Font family for the text, specified as a character vector or string scalar. Specify a font that appears in the font list in Microsoft PowerPoint. To see the font list, in PowerPoint, on the **Home** tab, in the **Font** group, click the arrow to the right of the font.

### **ComplexScriptFont** — Font family for complex scripts

`[]` (default) | character vector | string scalar

Font family for complex scripts, specified as a character vector or string scalar. Specify a font family to use when substituting in a locale that requires a complex script, such as Arabic or Asian, to render text.

### **FontColor** — Font color for text

`[]` (default) | character vector | string scalar

Font color for the text, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

### **FontSize — Font size of text**

[] (default) | character vector | string scalar

Font size of the text, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '11pt' specifies 11 points. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

### **Italic — Whether to use italic for text**

[] (default) | true | false

Whether to use italic for the text, specified as true or false. Specify true for italic text.

Data Types: logical

### **Strike — Strikethrough style for text**

[] (default) | 'single' | 'none' | 'double'

Strikethrough style for the text, specified as one of these values:

- 'single' — Single horizontal line
- 'none' — No strikethrough line
- 'double' — Double horizontal line

### **Subscript — Whether to display text as a subscript**

[] (default) | true | false

Whether to display the text as a subscript, specified as true or false. A setting of true renders the text as a subscript.

Data Types: logical

### **Superscript — Whether to display text as a superscript**

[] (default) | true | false

Whether to display the text as a superscript, specified as true or false. A setting of true renders the text as a superscript.

Data Types: logical

### **Underline — Underline style for text**

[] (default) | 'single' | 'double' | ...



Underline style for the text, specified as one of these values:

Value	Description
'single'	Single underline
'double'	Double underline
'heavy'	Thick underline
'words'	Only words underlined (not spaces)
'dotted'	Dotted underline
'dottedheavy'	Thick, dotted underline
'dash'	Dashed underline
'dashheavy'	Thick, dashed underline
'dashlong'	Long, dashed underline
'dashlongheavy'	Thick, long, dashed underline
'dotdash'	Dot-dash underline
'dotdashheavy'	Thick, dot-dash underline
'dotdotdash'	Dot-dot-dash underline
'dotdotdashheavy'	Thick, dot-dot-dash underline
'wavy'	Wavy underline
'wavyheavy'	Thick, wavy underline
'wavydouble'	Wavy, double underline
'none'	No underline

### **BackgroundColor** – Background color for text box placeholder

[] (default) | character vector | string scalar

Background color for the text box placeholder, specified as a character vector or string scalar that consists of a CSS color name or hexadecimal RGB value.

- For a list of CSS color names, see <https://www.w3.org/wiki/CSS/Properties/color/keywords>.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for the red, green, and blue values. For example, '#0000ff' specifies blue.

### **Valign** – Vertical alignment of text in the text box placeholder

[] (default) | 'top' | 'bottom' | ...

Vertical alignment of the title text in the text box placeholder, specified as a character vector or string scalar. Specify one of the values in the table.

Value	Description
'top'	Vertically aligned to the top of the text box placeholder
'bottom'	Vertically aligned to the bottom of the text box placeholder

Value	Description
'middle'	Vertically aligned to the middle of the text box placeholder
'topCentered'	Vertically aligned to the top and horizontally aligned to the center of the text box placeholder
'bottomCentered'	Vertically aligned to the bottom and horizontally aligned to the center of the text box placeholder
'middleCentered'	Vertically aligned to the middle and horizontally aligned to the center of the text box placeholder

**Name — Text box placeholder name**

character vector | string scalar

Text box placeholder name, specified as a character vector or string scalar.

**X — Upper left x-coordinate of position of text box placeholder**

character vector | string scalar

Upper left x-coordinate of the position of the text box placeholder in the slide, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '5in' specifies 5 inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Y — Upper left y-coordinate of position of text box placeholder**

character vector | string scalar

Upper left y-coordinate of the position of the text box placeholder in the slide, specified as a character vector or string scalar that consists of a number followed by a unit of measurement. For example, '5in' specifies 5 inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Width — Width of text box placeholder**

character vector | string scalar

Width of the text box placeholder, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '5in' specifies five inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

### Height — Height of text box placeholder

character vector | string scalar

Height of the text box placeholder, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '5in' specifies five inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

### Style — Text box placeholder formatting

cell array of PPT format objects

Text box placeholder formatting, specified as a cell array of PPT format objects. Formats that do not apply to a `TextBoxPlaceholder` object are ignored.

Add format objects by concatenating the existing value of the `Style` property with a cell array that contains the format objects that you are adding. For example:

```
import mlreportgen.ppt.*
slide = add(ppt,"Title Slide");
content = find(slide,"Title");
placeholderObj = content(1);
placeholderObj.Style = [placeholderObj.Style {Bold(true),FontColor("red")}];
replace(placeholderObj,'Test');
```

See “Presentation Formatting Approaches” on page 14-18.

### Children — Children of this PPT API object

cell array of PPT objects

Child elements of this object, specified as a cell array of PPT objects. This property is read-only.

### Parent — Parent of this PPT API object

PPT object

Parent of this object, specified as a PPT object. This property is read-only.

### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

**Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

**Methods****Public Methods**

`add` Add content to text box placeholder  
`replace` Replace text box placeholder content

**Examples****Replace Title Text in Placeholder**

Add a title slide to a presentation and then replace the slide placeholders for the title and subtitle with your title and subtitle text.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation.

```
ppt = Presentation("myTextBoxPlaceholderPresentation.pptx");  
open(ppt);
```

Add a title slide.

```
slide = add(ppt, "Title Slide");
```

Use the `find` method of the slide object to find the placeholder object that has the name `Title`.

```
titlePlaceholderObj = find(slide, "Title")
```

```
titlePlaceholderObj =  
    TextBoxPlaceholder with properties:
```

```
        Bold: []  
        Font: []  
ComplexScriptFont: []  
        FontColor: []  
        FontSize: []  
        Italic: []  
        Strike: []  
        Subscript: []  
        Superscript: []  
        Underline: []  
        BackgroundColor: []
```

```

VAlign: []
Name: 'Title'
X: '1524000emu'
Y: '1122363emu'
Width: '9144000emu'
Height: '2387600emu'
Style: []
Children: []
Parent: [1x1 mlreportgen.ppt.Slide]
Tag: 'ppt.TextBoxPlaceholder:21:61'
Id: '21:61'

```

The `find` method returns an `mlreportgen.ppt.TextBoxPlaceholder` object.

Replace the placeholder content with the title text.

```
replace(titlePlaceholderObj(1), "My Title");
```

Find the placeholder object for the subtitle.

```
subtitlePlaceholderObj = find(slide, "Subtitle")
```

```
subtitlePlaceholderObj =
    TextBoxPlaceholder with properties:
```

```

        Bold: []
        Font: []
ComplexScriptFont: []
        FontColor: []
        FontSize: []
        Italic: []
        Strike: []
        Subscript: []
        Superscript: []
        Underline: []
        BackgroundColor: []
        VAlign: []
        Name: 'Subtitle'
        X: '1524000emu'
        Y: '3602038emu'
        Width: '9144000emu'
        Height: '1655762emu'
        Style: []
        Children: []
        Parent: [1x1 mlreportgen.ppt.Slide]
        Tag: 'ppt.TextBoxPlaceholder:22:62'
        Id: '22:62'

```

The placeholder object for the subtitle is also an `mlreportgen.ppt.TextBoxPlaceholder` object.

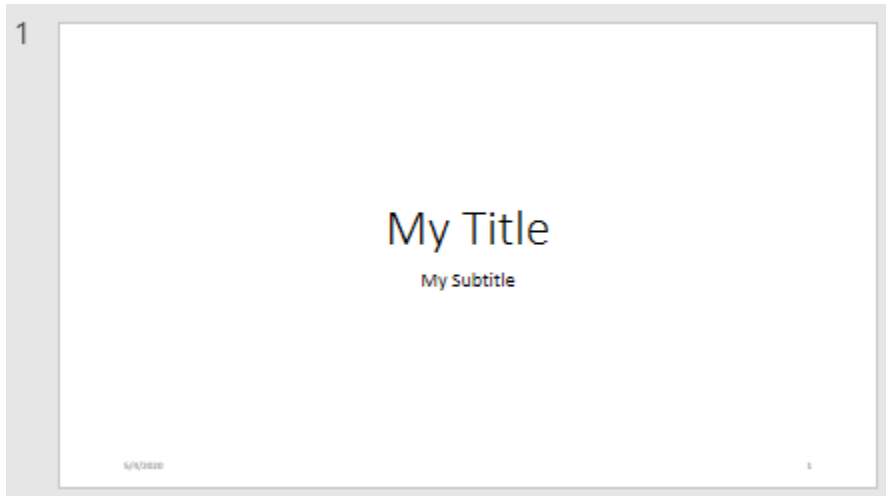
Replace the placeholder content with the text for the subtitle.

```
replace(subtitlePlaceholderObj(1), "My Subtitle");
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here is the title slide in the generated presentation:



### Add Paragraph to Slide Title

To add a paragraph to a slide title, use the `add` method of the `TextBoxPlaceholder` object that represents the title. This example creates a presentation with a `Title` and `Table` slide and then creates a second presentation from the first presentation. In the second presentation, the example adds content to the title of the `Title` and `Table` slide.

Import the PPT package so that you do not have to use long, fully qualified names for the PPT API classes.

```
import mlreportgen.ppt.*
```

Create a presentation and add a `Title` and `Table` slide to the presentation.

```
ppt = Presentation("myPPT1.pptx");
open(ppt);
slide = add(ppt, "Title and Table");
```

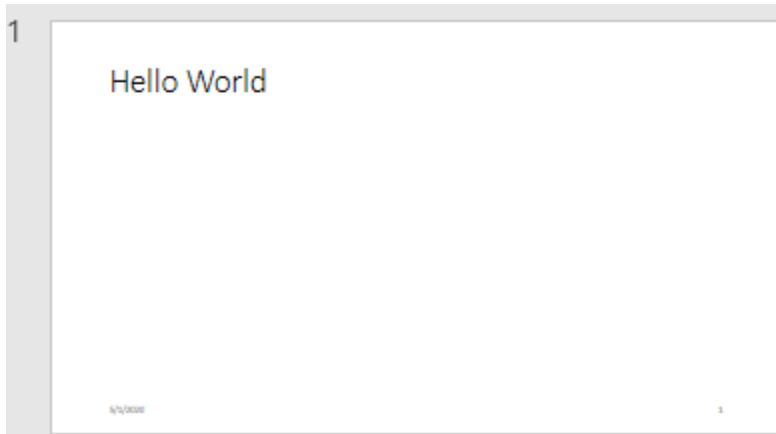
Find the text box placeholder for the title and replace the content.

```
contents = find(slide, "Title");
replace(contents(1), "Hello World");
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here is the title slide in the generated presentation:



Create a second presentation, using the first presentation as the template.

```
ppt = Presentation("myPPT2.pptx", "myPPT1.pptx");
open(ppt);
```

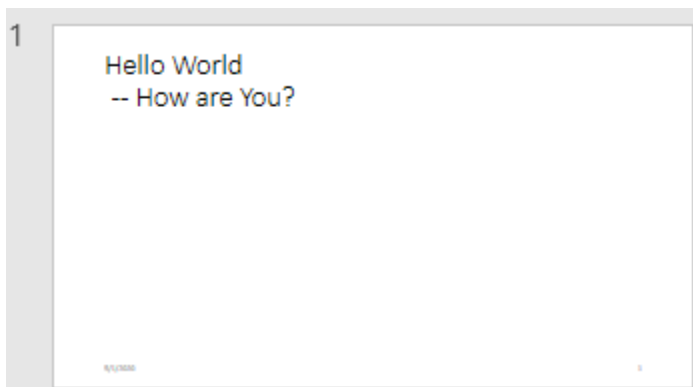
Find the text box placeholder for the title and add content to the title.

```
contents = find(ppt, "Title");
add(contents(1), "-- How are You?");
```

Close and view the presentation.

```
close(ppt);
rptview(ppt)
```

In the second presentation, the title has the content from the template presentation and the added content.



## Tips

- If you add a slide with a layout that has a placeholder for text that is not a title, the PPT API creates an `mlreportgen.ppt.BodyPlaceholder` or `mlreportgen.ppt.ContentPlaceholder` object.
- To see the placeholder objects that the PPT API creates for a slide object, view the `Children` property of the slide. For example, when you add a `Title` and `Content` slide to a presentation,

the Children property is an array that contains an `mlreportgen.ppt.TextBoxPlaceholder` object and an `mlreportgen.ppt.ContentPlaceholder` object.

```
ppt = mlreportgen.ppt.Presentation("test.pptx");
open(ppt);
slide = add(ppt,"Title and Content");
slide.Children(1)
```

```
ans =
```

```
TextBoxPlaceholder with properties:
```

```
    Bold: []
    Font: []
ComplexScriptFont: []
    FontColor: []
    FontSize: []
    Italic: []
    Strike: []
    Subscript: []
    Superscript: []
    Underline: []
    BackgroundColor: []
    VAlign: []
    Name: 'Title'
    X: []
    Y: []
    Width: []
    Height: []
    Style: []
    Children: []
    Parent: [1x1 mlreportgen.ppt.Slide]
    Tag: 'ppt.TextBoxPlaceholder:6:11'
    Id: '6:11'
```

```
slide.Children(2)
```

```
ans =
```

```
ContentPlaceholder with properties:
```

```
    Bold: []
    Font: []
ComplexScriptFont: []
    FontColor: []
    FontSize: []
    Italic: []
    Strike: []
    Subscript: []
    Superscript: []
    Underline: []
    BackgroundColor: []
    VAlign: []
    Name: 'Content'
    X: []
    Y: []
    Width: []
    Height: []
```



```
Style: []  
Children: []  
Parent: [1x1 mlreportgen.ppt.Slide]  
Tag: 'ppt.ContentPlaceholder:7:12'  
Id: '7:12'
```

## See Also

`mlreportgen.ppt.Paragraph` | `mlreportgen.ppt.PicturePlaceholder` |  
`mlreportgen.ppt.TablePlaceholder` | `mlreportgen.ppt.ContentPlaceholder` |  
`getLayoutNames`

## Topics

“Access PowerPoint Template Elements” on page 14-32

“Add and Replace Text” on page 14-59

## Introduced in R2015b

## mlreportgen.ppt.TextOrientation class

**Package:** mlreportgen.ppt

Orientation of text in a table entry

### Description

Use an mlreportgen.ppt.TextOrientation format object to specify the orientation of the text in a table entry in a PPT API presentation.

The mlreportgen.ppt.TextOrientation class is a handle class.

### Class Attributes

ConstructOnLoad	true
HandleCompatible	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

textOrientationObj = mlreportgen.ppt.TextOrientation() sets the “Value” on page 12-0 property to 'horizontal'.

textOrientationObj = mlreportgen.ppt.TextOrientation(orientation) sets the “Value” on page 12-0 property to orientation.

## Properties

### Value — Text orientation

'horizontal' (default) | 'down' | 'up'

Orientation of the table entry text, specified as a character vector or string scalar. Specify one of the values in the table.

Value	Description
'horizontal'	Text orientation is horizontal.
'down'	Text orientation is vertical, with the content rotated 90 degrees, clockwise.
'up'	Text orientation is vertical, with the content rotated 90 degrees, counterclockwise.

### Id — ID for this PPT API object

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

### Tag — Tag for this PPT API object

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## Examples

### Specify the Orientation of Text in a Table Entry in a Presentation

To specify the orientation of the text in a table entry in a presentation, add an `mlreportgen.ppt.TextOrientation` object to the `Style` property of the `mlreportgen.ppt.TableEntry` object that represents the table entry. This example generates a table with vertically oriented text in the first row.

Create a presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myTextOrientation.pptx");
open(ppt);
```

Add a slide to the presentation.

```
slide = add(ppt, "Title and Table");
```

Create a table.

```
t = Table({'Entry 11', 'Entry 12'; 'Entry 21', 'Entry 22'});
t.Height = '2in';
t.Width = '2in';
t.StyleName = "Medium style 2 - Accent 1";
t.Style = [t.Style {VAlign("middleCentered")}];
```

Specify the vertical text orientation for the entries in the first row.

```
tr1te1 = t.entry(1,1);
tr1te1.Style = [ tr1te1.Style {TextOrientation("down")} ];
tr1te2 = t.entry(1,2);
tr1te2.Style = [ tr1te2.Style {TextOrientation("down")} ];
```

Add a title and the table to the slide.

```
replace(slide, "Title", "Row One Text Has Vertical Orientation");
replace(slide, "Table", t);
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here is the table in the generated presentation:

Row One Text Has Vertical Orientation

Entry 11	Entry 12
Entry 21	Entry 22

12/3/2019 1

### **See Also**

`mlreportgen.ppt.Table` | `mlreportgen.ppt.TableEntry`

### **Topics**

“Create and Format Tables” on page 14-69

“Presentation Formatting Approaches” on page 14-18

“Presentation Format Inheritance” on page 14-21

### **Introduced in R2020a**

# mlreportgen.ppt.Underline class

**Package:** mlreportgen.ppt

Underline text

## Description

Format that underlines presentation text.

The mlreportgen.ppt.Underline class is a handle class.

## Class Attributes

HandleCompatible	true
ConstructOnLoad	true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`underlineObj = mlreportgen.ppt.Underline()` draws a single line under text.

`underlineObj = mlreportgen.ppt.Underline(style)` sets the Style property to the specified underline style.

## Properties

### Style – Underline style

'single' (default) | 'double' | ...

Underline style, specified as one of these values:

Value	Description
'single'	Single underline
'double'	Double underline
'heavy'	Thick underline
'words'	Only words underlined (not spaces)
'dotted'	Dotted underline
'dottedheavy'	Thick, dotted underline
'dash'	Dashed underline
'dashheavy'	Thick, dashed underline
'dashlong'	Long, dashed underline

Value	Description
'dashlongheavy'	Thick, long, dashed underline
'dotdash'	Dot-dash underline
'dotdashheavy'	Thick, dot-dash underline
'dotdotdash'	Dot-dot-dash underline
'dotdotdashheavy'	Thick, dot-dot-dash underline
'wavy'	Wavy underline
'wavyheavy'	Thick, wavy underline
'wavydouble'	Wavy, double underline
'none'	No underline

**Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

**Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

## Examples

**Underline Presentation Text**

Create a presentation.

```
import mlreportgen.ppt.*
```

```
ppt = Presentation('myUnderlinePresentation.pptx');
open(ppt);
slide = add(ppt, 'Title and Content');
```

Create a paragraph and append underlined text.

```
p = Paragraph('Hello World');

tWavy = Text(' wavy underline');
tWavy.Style = {Underline('wavy')};
append(p, tWavy);

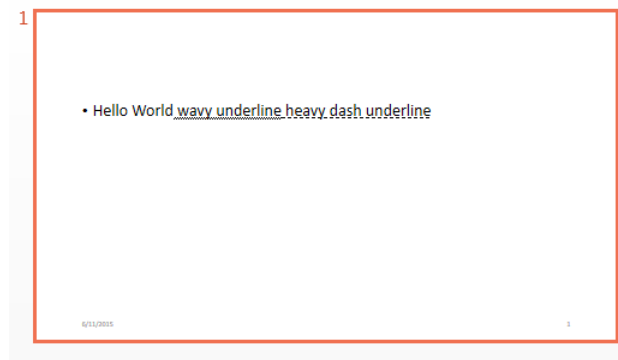
tDashed = Text(' heavy dash underline');
tDashed.Style = {Underline('dashheavy')};
append(p, tDashed);
```

Add the paragraph to the slide.

```
replace(slide, 'Content', p);
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```



## See Also

mlreportgen.ppt.Bold | mlreportgen.ppt.Italic

**Introduced in R2015b**

## mlreportgen.ppt.VAlign class

**Package:** mlreportgen.ppt

Vertical alignment of table entry content

### Description

Vertical alignment of table entry content.

### Construction

`vAlignObj = VAlign()` creates a vertical alignment object having the value 'top'.

`vAlignObj = VAlign(value)` creates a vertical alignment object having the specified value.

### Input Arguments

#### **value — Vertical alignment for table entry content**

'top' (default) | 'bottom' | 'middle' | 'topCentered' | 'middleCentered' | 'bottomCentered'

Vertical alignment for table entry content, specified as one of these values:

- 'top'
- 'bottom'
- 'middle'
- 'topCentered'
- 'middleCentered'
- 'bottomCentered'

### Output Arguments

#### **vAlignObj — Vertical alignment of table entry content**

mlreportgen.ppt.VAlign object

Vertical alignment of table entry content, returned as an mlreportgen.ppt.VAlign object.

### Properties

#### **Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

#### **Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.



The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

### Value — Vertical alignment of table entry content

```
'top' (default) | 'bottom' | 'middle' | 'topCentered' | 'middleCentered' |
'bottomCentered'
```

Vertical alignment of table entry content, specified as a character vector.

## Examples

### Add a Table

Create a presentation.

```
import mlreportgen.ppt.*

ppt = Presentation('myVAlign.pptx');
open(ppt);
slide1 = add(ppt, 'Title and Content');
```

Create a table using a cell array. Set the vertical alignment for each entry.

```
table1 = Table();
row1 = TableRow();
p1 = Paragraph('top');
r1e1 = TableEntry();
r1e1.Style = {VAlign('top'),FontSize('.5in')};
append(r1e1,p1);
append(row1,r1e1);

p2 = Paragraph('middle');
r1e2 = TableEntry();
r1e2.Style = {VAlign('middle')};
append(r1e2,p2);
append(row1,r1e2);

row2 = TableRow();
p3 = Paragraph('bottom');
r2e1 = TableEntry();
r2e1.Style = {VAlign('bottom')};
append(r2e1,p3);
append(row2,r2e1);

p4 = Paragraph('middle centered');
r2e2 = TableEntry();
r2e2.Style = {VAlign('middleCentered'),FontSize('.5in')};
append(r2e2,p4);
append(row2,r2e2);

append(table1,row1);
append(table1,row2);
```

Add the table to the slide, generate the presentation, and open the myVAlign presentation.

```
replace(slide1, 'Content', table1);  
close(ppt);  
rptview(ppt);
```

1

top	middle
bottom	middle centered

### See Also

`mlreportgen.ppt.TableEntry` | `mlreportgen.ppt.HAlign`

### Topics

“Create and Format Tables” on page 14-69

“Presentation Formatting Approaches” on page 14-18

**Introduced in R2015b**

# mlreportgen.ppt.WarningMessage class

**Package:** mlreportgen.ppt

Warning message

## Description

Create a warning message with the specified text originating from the specified source object.

## Construction

`warningMsgObj = WarningMessage(text,source)` creates a warning message with the specified text originating from the specified source object.

### Input Arguments

**text — Message text**

character vector

The text to display for the message, specified as a character vector.

**source — PPT object from which message originates**

a PPT object

The PPT object from which the message originates, specified as a PPT object.

### Output Arguments

**warningMsgObj — Warning message**

mlreportgen.ppt.WarningMessage object

Warning message, returned as an mlreportgen.ppt.WarningMessage object.

## Properties

**Id — ID for this PPT API object**

character vector | string scalar

ID for this PPT API object, specified as a character vector or string scalar. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

**Source — Source PPT object from which message originates**

a PPT object

Source PPT object from which the message originates, specified as a PPT object.

**Tag — Tag for this PPT API object**

character vector | string scalar

Tag for this PPT API object, specified as a character vector or string scalar. A session-unique tag is generated as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the object.

Specifying your own tag value can help you to identify where an issue occurred during presentation generation.

### Text — Text of message

character vector

Message text, specified as a character vector.

## Methods

Use WarningMessage methods similar to how you use ProgressMessage methods.

Method	Purpose
formatAsHTML	Wrap message in HTML tags.
formatAsText	Format message as text.
passesFilter	Determine if message passes filter.

## Examples

### Create a Warning Message

```
import mlreportgen.ppt.*;
pre = Presentation('myPresentation.pptx');

dispatcher = MessageDispatcher.getTheDispatcher;
l = addlistener(dispatcher,'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsText));

open(pre);
dispatch(dispatcher,WarningMessage('invalid slide',pre));

titleText = Text('This is a Title');
titleText.Style = {Bold};

replace(pre,'Title',titleText);

close(pre);
```

Delete the listener to avoid duplicate reporting of message objects during a MATLAB session.

```
delete(l);
```

### See Also

dispatch | mlreportgen.ppt.MessageEventData

### Topics

“Display Presentation Generation Messages” on page 14-14

**Introduced in R2015b**

## mlreportgen.report.Axes class

**Package:** mlreportgen.report

Axes reporter

### Description

Use objects of the mlreportgen.report.Axes class to report on axes found in a MATLAB figure.

The mlreportgen.report.Axes class is a handle class.

### Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`axes = mlreportgen.report.Axes()` creates an Axes object with default property values. You must specify the axes handle to report by setting the Source property. Use other properties to specify report options.

---

**Note** The figure containing the axes must remain open until you add the axes reporter to a report.

---

`axes = mlreportgen.report.Axes(source)` creates a reporter that adds the axes specified by source and sets the Source property to source.

`axes = mlreportgen.report.Axes(Name=Value)` sets properties by using name-value arguments. Specify multiple name-value arguments in any order.

## Properties

### Snapshot — Axes image properties

mlreportgen.report.FormalImage object

Properties for the axes image, specified as an mlreportgen.report.FormalImage object. Use the properties of the FormalImage object to specify the size of a snapshot if you cannot use Scaling, Height and Width properties without cropping the image.

---

**Note** The axes reporter initializes the Snapshot property. Do not reset this property.

---

### Source — Axes image source

axes handle

Axes image source, specified as an axes handle.

### SnapshotFormat — Snapshot image format

'svg' (default) | 'bmp' | 'jpg' | 'png' | 'emf' | 'tif' | 'pdf'

Snapshot image format, specified as a character vector or string scalar. Supported formats are:

- 'svg' — Scalable Vector Graphics.
- 'bmp' — Bitmap image.
- 'jpg' — JPEG image.
- 'png' — PNG image.
- 'emf' — Enhanced metafile. This format is supported only in DOCX output on Windows platforms.
- 'tif' — Tag Image File format. This format is not supported in HTML reports.
- 'pdf' — PDF image. This format is supported only in PDF reports.

See “Compatibility Considerations” on page 12-811.

### Scaling — Scaling options for axes image

'auto' (default) | 'custom' | 'none'

Scaling options for the axes image, specified as 'auto', 'custom', or 'none'.

- 'auto' — For PDF or Word reports, this setting scales the axes image to fit the current page layout while maintaining its aspect ratio. First, the axes image scales to the page width. If the image height exceeds the page height, the image scales down again. This additional scaling ensures that the image fits the current page with an extra one inch spacing. This setting does not apply to HTML reports.
- 'custom' — This setting scales the axes image based on the values of the Height and Width properties.
- 'none' — The image is not scaled.

---

**Note** The 'auto' and 'custom' options use the MATLAB print command to resize the figure. If the figure is too large to fit legibly in the specified space, the print command crops the snapshot image. To avoid cropping, you can set Scaling to 'none' and use the reporter specified by the Snapshot property to size the axes image. Because this reporter reduces the size of the text with the rest of the image, a user might need to zoom the image to discern fine detail. See “Resize Figure Snapshot Image” on page 12-808.

---

### Height — Height of snapshot image

character vector | string scalar

Height of snapshot image, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '2in' specifies two inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches

- mm — millimeters
- pc — picas
- pt — points

Example: '2in'

**Width — Width of snapshot image**

character vector | string scalar

Width of snapshot image, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '2in' specifies two inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

Example: '3in'

**PreserveBackgroundColor — Preserve axes background color**

false (default) | true

Preserve the axes background color in the snapshot, specified as `true` or `false`. If `PreserveBackgroundColor` is `true`, the background color of the snapshot is the same as the background color of the axes. If `PreserveBackgroundColor` is `false`, the background color of the axes image is white.

**TemplateSrc — Source of template for this reporter**

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

**TemplateName — Name of template for this reporter**

character vector | string scalar



Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (TemplateSrc) for this reporter.

### LinkTarget — Hyperlink target for this reporter

[] (default) | character vector | string scalar | mlreportgen.dom.LinkTarget object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an mlreportgen.dom.LinkTarget object. A character vector or string scalar value is converted to a LinkTarget object. The link target immediately precedes the content of this reporter in the output report.

## Methods

### Public Methods

getSnapshotImage	Get axes image path
mlreportgen.report.Axes.createTemplate	Create axes reporter template
mlreportgen.report.Axes.customizeReporter	Create class derived from axes reporter class
mlreportgen.report.Axes.getClassFolder	Get location of folder containing mlreportgen.report.Axes class definition file
copy	Create copy of reporter object and make deep copies of certain property values
getImpl	Get implementation of reporter

## Examples

### Add an Axes Image to Report

This example shows how to add an axes image to a report and set the axes image caption and dimensions.

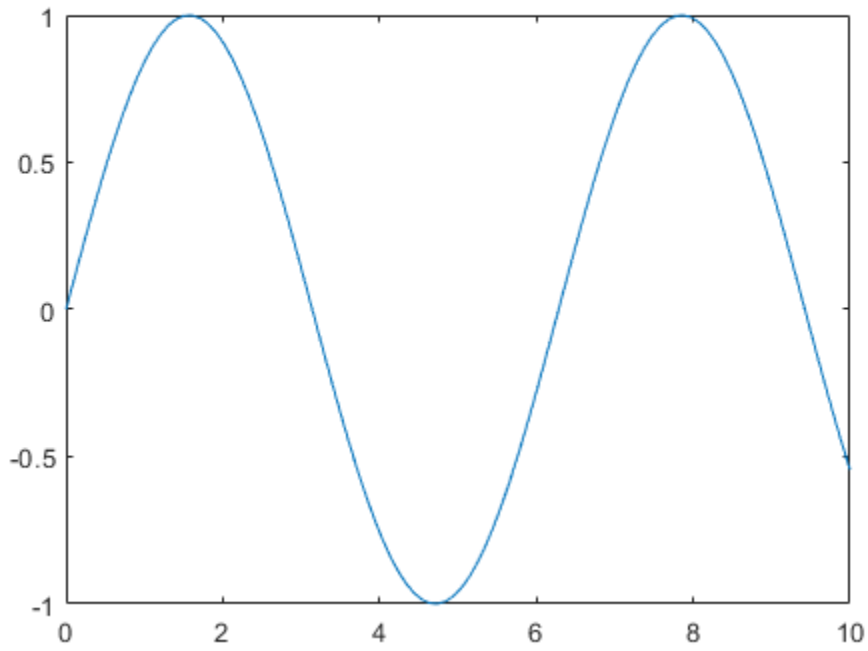
```
import mlreportgen.report.*
```

Create a PDF report and chapter.

```
rpt = Report("Add Axis Figure", "pdf");
chapter = Chapter("Axes");
```

Create data and a set of axes for the plot.

```
ax = axes(Position=[0.1 0.1 0.7 0.7]);
x1 = linspace(0,10,100);
y1 = sin(x1);
plot(ax,x1,y1);
```



Create an axes reporter by using `mlreportgen.report.Axes`.

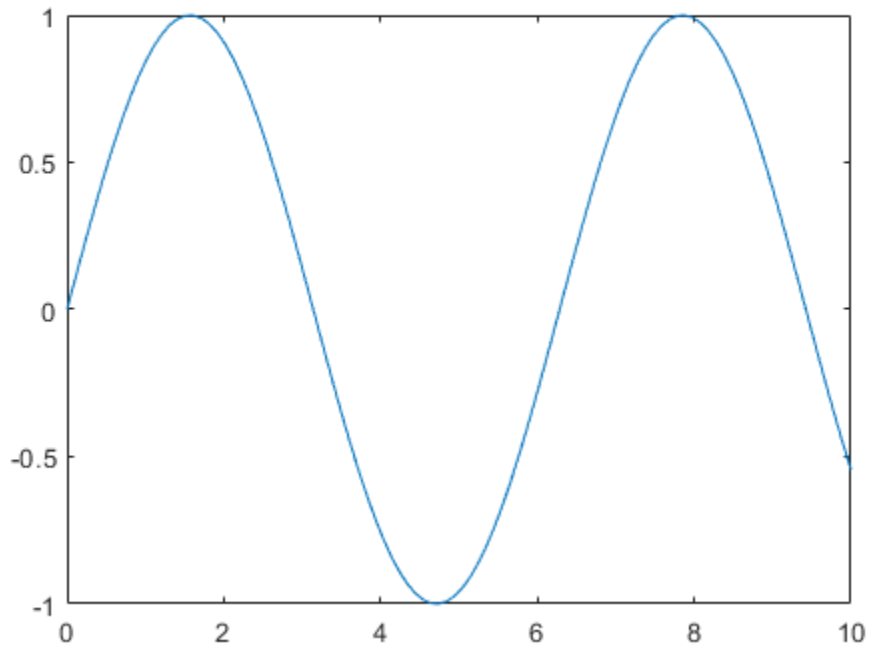
```
axesRpt = Axes(ax);
```

Use the custom scaling options to set the height and width of the image to be four inches.

```
axesRpt.Scaling = "custom";  
axesRpt.Height = "4in";  
axesRpt.Width = "4in";
```

Add the axes reporter to the chapter and the chapter to the report.

```
append(chapter, axesRpt);  
add(rpt, chapter);
```



Close the report and view it.

```
close(rpt);  
rptview(rpt);
```

### See Also

[mlreportgen.finder.AxesFinder](#) | [mlreportgen.report.Figure](#) | [mlreportgen.finder.AxesResult](#)

**Introduced in R2021b**

## mlreportgen.report.BaseTable class

**Package:** mlreportgen.report mlreportgen.report mlreportgen.report

**Superclasses:** mlreportgen.report.Reporter

Create table reporter

### Description

Use an object of the mlreportgen.report.BaseTable class to create a reporter for a table that has a title.

The mlreportgen.report.BaseTable class is a handle class.

### Class Attributes

HandleCompatible true

## Creation

### Description

`table = mlreportgen.report.BaseTable()` creates an empty table reporter. Use the reporter properties to specify the table content, title, style, and width.

`table = mlreportgen.report.BaseTable(content)` creates a table reporter and sets the Content property to content.

`table = mlreportgen.report.BaseTable(Name=Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order.

## Properties

### Title – Table title

string scalar | character vector | ...

Table title, specified as one of these values:

- String scalar or character vector
- DOM object
- 1-by-*N* or *N*-by-1 array of strings or DOM objects
- 1-by-*N* or *N*-by-1 cell array of strings, character vectors, or DOM objects
- Hole reporter returned by the getTitleReporter method

If the title value is inline content, which is content included in a paragraph, the table reporter uses a template stored in its template library to format the title. The template automatically numbers the table title using a format that depends on whether the table is in a numbered or unnumbered chapter.

- A table in a numbered chapter has a title text prefix of the form "Table *N.M*" where *N* is the chapter number and *M* is the table number in the chapter. For example, the prefix for the third table in the second chapter of the report is Table 2.3.
- A table in unnumbered chapter has a title text prefix of the form "Table *N*" where *N* is 1 for the first table in the report, 2 for the second table, and so on.

In many non-English locales, the title prefix is translated to the language and format of the locale. See the `Locale` property of `mlreportgen.report.Report` for a list of translated locales.

**Attributes:**

```
GetAccess          public
SetAccess          public
```

**Content — Table content**

```
mlreportgen.dom.Table object | mlreportgen.dom.FormalTable object |
mlreportgen.dom.MATLABTable object | ...
```

Table content, specified as one of these values:

- `mlreportgen.dom.Table` object
- `mlreportgen.dom.FormalTable` object
- `mlreportgen.dom.MATLABTable` object
- Two-dimensional array or cell array of DOM or built-in MATLAB objects
- Hole reporter returned by `getContentReporter` method

Use the `BaseTable` constructor or `add` method to set this property. You cannot set it directly.

**Attributes:**

```
GetAccess          public
SetAccess          public
```

**TableStyleName — Style to apply to table**

```
string scalar | character vector
```

Name of the style to apply to the table, specified as a string scalar or character vector. The specified style must be a table style defined in the template used by the report to which you append this table or in the template of a reporter added to the report.

If `TableStyleName` is empty, the table style is the default table style defined by the template of the reporter, which is a grid.

**Attributes:**

```
GetAccess          public
SetAccess          public
```

**TableWidth — Width of the table**

```
[] | string scalar | character vector
```

Width of this table, specified as a string scalar or character vector that consists of a number followed by an abbreviation of a unit of measurement. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

**Attributes:**

GetAccess	public
SetAccess	public

**MaxCols — Maximum number of columns to display per table slice**

Inf (default) | positive integer

Maximum number of columns to display per table slice, specified as `Inf` or as a positive integer. If the value of this property is `Inf`, all original table columns are included in a single table. A `MaxCols` value greater than or equal to the number of table columns also produces a single table with all columns. Large table data sets may be illegible.. Set this property to the number of columns from the original table that fit legibly on a page. To determine an optimal value, iterate setting the `MaxCols` value and viewing the report.

**Attributes:**

GetAccess	public
SetAccess	public

**RepeatCols — Number of initial columns to repeat per slice**

0 (default) | positive integer

Number of initial columns to repeat per slice, specified as 0 or a positive integer. A nonzero number, *n*, repeats the first *n* columns of the original table in each slice. The `MaxCols` property value includes the `RepeatCols` property value. For example, if `MaxCols` is 6 and `RepeatCols` is 2, each table slice has a total of six columns with the first two columns repeated from the original table.

**Attributes:**

GetAccess	public
SetAccess	public

**TableSliceTitleStyleName — Name of style applied to sliced table title**

[] (default) | string scalar | character vector

Name of the custom style to apply to the titles of table slices, specified as a string or a character vector.. The specified style must be defined in the report to which this reporter is added. If this property is empty ('', "", or []), the slice titles use the default style defined in the reporter template.

**Attributes:**

GetAccess	public
SetAccess	public

**TemplateSrc — Source of template for this reporter**

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template specified by the `TemplateSrc` property of this reporter.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**LinkTarget — Hyperlink target for this reporter**

[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**TableEntryUpdateFcn — Update table entries callback**

[] (default) | function handle

Update table entries callback, specified as a function handle. The callback function accepts an `mlreportgen.dom.TableEntry` object as the input argument. Appending the `BaseTable` object to a report triggers the callback function once for each table entry before the table is appended to the report. Use this function to customize the appearance of table entries based on the content of the entries. For example, see “Customize Table Entries by Content in a PDF Report” on page 12-785.

**Attributes:**

GetAccess	public
SetAccess	public

**Methods****Public Methods**

Method	Purpose
<code>m1reportgen.report.BaseTable.createTemplate</code>	Create table template
<code>m1reportgen.report.BaseTable.customizeReporter</code>	Create custom base table reported class
<code>m1reportgen.report.BaseTable.getClassFolder</code>	Get base table class definition file location
<code>getContentReporter</code>	Get base table content hole reporter
<code>getTitleReporter</code>	Get base table title reporter
<code>getImpl</code>	Get implementation of reporter
<code>copy</code>	Create copy of reporter object and make deep copies of property values that reference a reporter, ReporterLayout, or DOM object

**Examples****Add Tables with Titles to a Report**

Add two tables that have titles to a report. The first table is a rank 5 magic square. The second table includes two images.

```
import m1reportgen.report.*
import m1reportgen.dom.*

rpt = Report("tables");
chapter = Chapter();
chapter.Title = "Table example";
add(rpt,chapter);

table = BaseTable(magic(5));
table.Title = "Rank 5 Magic Square";
add(rpt,table);

add(rpt,Paragraph());
imgSize = {Height("2in"),Width("2in")};
img1 = Image(which("b747.jpg"));
img1.Style = imgSize;
img2 = Image(which("peppers.png"));
img2.Style = imgSize;
table = BaseTable({"Boeing 747" "Peppers"; img1, img2});
table.Title = "Picture Gallery";
```



```
add(rpt, table);
delete(gcf);
rptview(rpt);
```

## Chapter 1. Table example

**Table 1.1. Rank 5 Magic Square**

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

**Table 1.2. Picture Gallery**

Boeing 747	Peppers
	

### Customize Table Entries by Content in a PDF Report

This example shows how to customize table entries, based on the content of the entries when using an `mlreportgen.report.BaseTable` reporter. You can run this example only on MATLAB® version R2022a or later.

Import these packages, so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
import mlreportgen.report.*
```

Create a table with student names and grades.

```
Student_names = ["Charlie", "Sarah", "John", "Teena", "Alfred", "Emma", "Bill", "Tyler"]';
Student_grades = [83, 42, 75, 98, 78, 49, 91, 88]';
studentsTable = FormalTable(["Student Name", "Student Grade"], [Student_names, Student_grades]);
```

```
studentsTable.Header.Style{end+1} = BackgroundColor("silver");
studentsTable.Width = "250pt";
```

Create an `mlreportgen.dom.BaseTable` reporter with the students table.

```
baseTabReporter = BaseTable(studentsTable);
```

Assign a handle of the callback function `tabEntryUpdateCB`, that is defined in the file `tabEntryUpdateCB.mlx`, to the `TableEntryUpdateFcn` property of the reporter.

Use the command `edit tabEntryUpdateCB` in the command line to see the code of the callback function.

```
baseTabReporter.TableEntryUpdateFcn = @tabEntryUpdateCB;
```

Create an `mlreportgen.dom.Report` of type PDF, then append the base table reporter to the report.

```
document = Report("Update_table_entries_example_report","pdf");
append(document,baseTabReporter);
```

Close and view the report.

```
close(document);
rptview(document);
```

Student Name	Student Grade
Charlie	83
Sarah	42
John	75
Teena	98
Alfred	78
Emma	49
Bill	91
Tyler	88

## See Also

`mlreportgen.report.Reporter` | `mlreportgen.report.Report` | `mlreportgen.dom.Table` | `mlreportgen.dom.FormatTable` | `mlreportgen.dom.MATLABTable` | `mlreportgen.dom.TableEntry`

## Topics

“What Is a Reporter?” on page 1-3

**Introduced in R2017b**

## mlreportgen.report.BlockContent class

**Package:** mlreportgen.report

Fill block content hole reporter

### Description

Reporter for templates with a block content hole. A block hole is a hole that can contain paragraphs or any other type of content. Other reporters use this `BlockContent` reporter to fill block holes in their templates. Reporters have methods that return instances of this object. Using these object instances, you can customize the format of the content used to fill the block holes in their templates. For example, for the `BaseTable` reporter, the `getContentReporter` method returns the instance that the `BaseTable` reporter uses to fill the `Content` hole in its template. To customize the content format, specify a custom template for the `BlockContent` reporter returned by the `getContentReporter` method.

---

**Note** Reporters create instances of this object. You do not need to create this object yourself.

---

### Properties

#### HoleId — ID of hole

string

ID of hole to be filled by this reporter, specified as a string.

#### Content — Content of hole

string | character array | ...

Content of hole to be filled by this reporter, specified as one of these values:

- String or character array
- DOM object
- 1-by-*N* or *N*-by-1 array of strings or DOM objects
- 1-by-*N* or *N*-by-1 cell array of strings, character arrays, and/or DOM objects

#### TemplateSrc — Source of template for this reporter

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the

TemplateSrc property is empty, this reporter uses the default reporter template for the output type of the report.

**TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (TemplateSrc) for this reporter.

**LinkTarget — Hyperlink target for this reporter**

[] (default) | character vector | string scalar | mlreportgen.dom.LinkTarget object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an mlreportgen.dom.LinkTarget object. A character vector or string scalar value is converted to a LinkTarget object. The link target immediately precedes the content of this reporter in the output report.

**See Also**

mlreportgen.report.Report | mlreportgen.report.Reporter

**Topics**

“What Is a Reporter?” on page 1-3

**Introduced in R2018b**

## mlreportgen.report.Chapter class

**Package:** mlreportgen.report

**Superclasses:** mlreportgen.report.Section

Chapter reporter

### Description

Use an object of the mlreportgen.report.Chapter class to add a chapter to a report.

The mlreportgen.report.Chapter class is a handle class.

### Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

ch = mlreportgen.report.Chapter() creates a reporter that generates a chapter. The chapter has a new page layout defined by the default template of the reporter.

The default template is a portrait page with a header and a footer. The header is empty. If a chapter is the first chapter of the report, the footer contains an automatically generated page number that starts with 1. If the chapter is not the first chapter, the page numbering continues from the last page of the previous chapter. Use the Layout property to override some of the page layout features of the chapter, such as the orientation.

Use the Title property to specify the title.

To add content to the chapter, use the append method of the mlreportgen.report.Chapter object.

---

**Note** Before you add a chapter to a report, add all of the content to the chapter. Once you add the chapter to a report, you cannot add more content to the chapter.

---

ch = mlreportgen.report.Chapter(title) creates a chapter with the specified title. See the Title property.

ch = mlreportgen.report.Chapter(Name=Value) sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order.

## Properties

### Title — Chapter title

string scalar | character vector | DOM object | ...

Chapter title, specified as one of these values:

- String scalar or character vector
- DOM object
- 1-by-*N* or *N*-by-1 array of string scalars or DOM objects
- 1-by-*N* or *N*-by-1 cell array of string scalars, character vectors, and/or DOM objects
- `SectionTitle` reporter returned by the `getTitleReporter` method

The title appears at the beginning of the chapter and in the header of all chapter pages except the first page. The title also appears in the table of contents of the report.

Inline objects are objects that a paragraph can contain. If the title value is an inline object, the `Chapter` object uses a template from its template library to create a title based on the value. The template used to create the title depends on whether the title is numbered. A chapter title is numbered by default. To turn off numbering for this chapter, use the `Numbered` property. To turn off numbering for all other chapters in the report, use the `mlreportgen.report.Section.number` method.

If the title is numbered, the title is prefixed in English reports by a string of the form `Chapter N`, where *N* is the automatically generated chapter number. In some other locales, the English prefix is translated to the language of the locale. See the `Locale` property of `mlreportgen.report.Report` for a list of translated locales.

You can use inline DOM objects to override the character formatting specified by the chapter default title templates.

If the title value is a DOM paragraph or other DOM block object, the chapter inserts the object at the beginning of the chapter. If you use a DOM block object, you can use block elements to customize the spacing, alignment, and other properties of the chapter title. In this case, you must fully specify the title format and provide title numbering yourself.

---

**Note** For the page headers to correctly display the title, the style name of the title must be `SectionTitle1`. If the title is specified as a DOM `mlreportgen.dom.Paragraph` object with no `StyleName` set, the `StyleName` property is automatically changed to the correct style name. You can customize the paragraph style by including DOM style objects in the `Style` property of the `Paragraph` object. If you use a custom template to format the title, make sure the style name used by the template is `SectionTitle1`. Customize the title style by modifying the `SectionTitle1` style in the custom template.

---

#### Attributes:

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

### Numbered — Whether to number this chapter

`true` | `false`

Whether to number this chapter, specified as `true` or `false`. If the value of `Numbered` is empty or `true`, the chapter is numbered relative to other chapters in the report. The chapter number appears in the title. If the value of `Numbered` is `false`, this chapter is not numbered. The value of the `Numbered` property overrides the numbering specified for all report chapters by the `mlreportgen.report.Chapter.number` method.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**Content — Content of this section**

`string scalar` | `character vector` | `inline object` | ...

Content of the section, specified as one of these values:

- String scalar or character vector
- DOM objects that can be added to a DOM document part
- Reporters, including `Section` reporters
- `1xN` or `Nx1` array of string scalars or character vectors
- `1xN` or `Nx1` cell array of string scalars, character vectors, and/or DOM objects

Use the `Chapter` constructor or `append` method to set this property. You cannot set it directly.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>Restricts access</code>

**Layout — Layout of this chapter**

`mlreportgen.report.ReporterLayout`

Layout of this chapter, specified as an `mlreportgen.report.ReporterLayout` object. Use the properties of the `ReporterLayout` object to override some of the chapter layout properties that are defined in the template for the chapter.

Example: `chapter.Layout.Landscape = true`

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>Restricts access</code>

**TemplateSrc — Source of template for this reporter**

`[]` (default) | `character vector` | `string scalar` | `reporter` or `report` | `DOM document` or `document part`

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- `Reporter` or `report` whose template is used for this reporter or whose template library contains the template for this reporter
- `DOM document` or `document part` whose template is used for this reporter or whose template library contains the template for this reporter



The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

**Attributes:**

`GetAccess` public  
`SetAccess` public

**TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template specified by the `TemplateSrc` property of this reporter.

**Attributes:**

`GetAccess` public  
`SetAccess` public

**LinkTarget — Hyperlink target for this reporter**

[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

**Attributes:**

`GetAccess` public  
`SetAccess` public

## Methods

### Public Methods

<code>append</code>	Add content to chapter
<code>add</code>	(Not recommended) Add content to chapter
<code>getTitleReporter</code>	Get chapter title reporter
<code>mlreportgen.report.Chapter.number</code>	<code>mlreportgen.report.Chapter.number(reprt,numbering)</code> specifies whether to number chapters in a report. If <code>numbering</code> is <code>true</code> , all chapters in the report are numbered.

<code>mlreportgen.report.Chapter.createTemplate</code>	<code>mlreportgen.report.Chapter.createTemplate(templatePath,type)</code> creates a copy of the Chapter template specified by <code>type</code> at the location specified by <code>templatePath</code> . You can use this method to create a copy of a default Section template to serve as a starting point for creating your own custom template.
<code>mlreportgen.report.Chapter.customizeReporter</code>	<code>mlreportgen.report.Chapter.customizeReporter(toClasspath)</code> creates an empty class derived from the Chapter reporter class with the name <code>toClasspath</code> . You can use the generated class as a starting point for creating a custom version of the Chapter reporter.
<code>mlreportgen.report.Chapter.getClassFolder</code>	<code>mlreportgen.report.Chapter.getClassFolder()</code> returns the path of the folder that contains the class definition file for the Chapter reporter class.
<code>copy</code>	Create copy of reporter object and make deep copies of property values that reference a reporter, ReporterLayout, or DOM object
<code>getImpl</code>	Get DOM implementation for this reporter

## Examples

### Add a Chapter to a Report

Add a section to a chapter and the chapter to a report. Set the layout orientation of the chapter to landscape.

```
import mlreportgen.report.*
import mlreportgen.dom.*

rpt = Report('My Report','pdf');

append(rpt,TitlePage(Title='My Report'));

chapter = Chapter('Images');
chapter.Layout.Landscape = true;
append(chapter,Section(Title='Boeing 747',Content=Image('b747.jpg')));

append(rpt,chapter);
close(rpt);
rptview(rpt);
```

Here is the first page of Chapter 1 in the generated report.

## Chapter 1. Images

### 1.1. Boeing 747



---

1

## Compatibility Considerations

### **add method is not recommended**

*Not recommended starting in R2020b*

Starting in R2020b, use the `append` method instead of the `add` method to add content to objects of these Report API classes:

- `mlreportgen.report.Report`
- `mlreportgen.report.Chapter`
- `mlreportgen.report.Section`

To add content to a DOM API object, such as an `mlreportgen.dom.Paragraph` object, continue to use the `append` method of the DOM object. The advantage of using `append` to add content to Report API objects is that you use the same method name as you use to add content to DOM API objects.

There are no plans to remove the add methods of the Report, Chapter, or Section classes. Report API programs that use the add methods will continue to run.

To update existing code, replace the method name add with append as shown by the examples in the table.

Not Recommended	Recommended
<pre>import mlreportgen.report.* import mlreportgen.dom.*  rpt = Report("My Report","pdf"); ch = Chapter("My Chapter"); sect = Section("My Section"); para = Paragraph("My Content "); append(para,"more Content"); add(sect,para); add(ch,sect); add(rpt,ch);  close(rpt); rptview(rpt);</pre>	<pre>import mlreportgen.report.* import mlreportgen.dom.*  rpt = Report("My Report","pdf"); ch = Chapter("My Chapter"); sect = Section("My Section"); para = Paragraph("My Content "); append(para,"more Content"); append(sect,para); append(ch,sect); append(rpt,ch);  close(rpt); rptview(rpt);</pre>

## See Also

mlreportgen.report.Section | mlreportgen.report.Report |  
mlreportgen.report.Reporter | mlreportgen.report.ReportLayout

## Topics

“What Is a Reporter?” on page 1-3

## Introduced in R2017b

# mlreportgen.report.Equation class

**Package:** mlreportgen.report

**Superclasses:** mlreportgen.report.Reporter

Equation reporter

## Description

Create an equation reporter that adds an equation to a report.

An equation is added to a report as an image of the formatted equation. By default, the image is embedded in an empty, centered paragraph. Optionally, it can be appended in line with other text in a paragraph. The image, and therefore, the equation can be scaled to any size.

The snapshot image of the equation is stored in the temporary folder of the report. When the report is closed, the equation image is copied into the report and, then, the image is deleted from the temporary folder. To prevent the equation image files from being deleted, use the `Debug` property of the report. See `mlreportgen.report.Report`.

The `mlreportgen.report.Equation` class is a `handle` class.

## Class Attributes

`HandleCompatible` true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`equation = mlreportgen.report.Equation()` creates an empty equation reporter object. Use the object properties to specify the equation and its formatting.

`equation = mlreportgen.report.Equation(markup)` formats the equation that is specified by the LaTeX markup for the equation. See the `Content` property.

`equation = mlreportgen.report.Equation(Name=Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order.

## Properties

### Content — LaTeX markup

`[]` (default) | string scalar | character vector

LaTeX markup for the equation, specified as a string scalar or character vector. You can use any LaTeX markup that is supported by the `Interpreter` property of a MATLAB text object. See `Text`.

**Attributes:**

GetAccess public  
SetAccess public

**FontSize — Font size**

[] (default) | positive integer

Font size for the formatted equation, specified as empty, or as a positive integer. If the `FontSize` property is empty, the font size defaults to 10.

**Attributes:**

GetAccess public  
SetAccess public

**Color — Font color**

[] (default) | character vector | string scalar

Font color, specified as a character vector or string scalar that contains the color name or equivalent hexadecimal or RGB triplet specification.

The color specification depends on the value of the `UseDirectRenderer` property. If the property value is `true`, specify a valid CSS color name, RGB triplet, or hexadecimal value. See <https://www.w3.org/TR/2018/REC-css-color-3-20180619/>. If the property value is `false`, specify the color by using a name, short name, RGB triplet, or hexadecimal value as described in the `Color` property of `Text Properties`. This table shows how to specify the color red for different values of the `UseDirectRenderer` property.

UseDirectRenderer Value	Name	RGB Triplet	Hexadecimal Code
true	"red"	"rgb(255,0,0)"	"#FF0000"
false	"red" or "r"	"[1,0,0]"	"#FF0000"

**Attributes:**

GetAccess public  
SetAccess public

**BackgroundColor — Name of background color**

[] (default) | string scalar | character vector

Name of background color for the formatted equation, specified as empty, or as a string scalar or character vector. If `Color` is empty, the background color defaults to white. You can use long or short color names. For a list of valid color names, see the `Color` property in `Text`.

If the `UseDirectRenderer` property is `true`, the `BackgroundColor` property is ignored.

**Attributes:**

GetAccess public  
SetAccess public

**DisplayInline — Display equation in line with text**

false (default) | true

Display equation in line with text, specified as `true` or `false`.

If the `DisplayInline` property is set to `false`, the reporter takes an image of the equation, wraps the image in a paragraph, and adds the paragraph to the report. In the report, the equation is on a line by itself. See “Add an Equation to a Chapter” on page 12-801. Use this option to fill block holes in a template.

If the `DisplayInline` property is set to `true`, the equation image is not wrapped in a paragraph. To add the equation to the report, get the equation image by using the `getImpl` method and then add the image to a paragraph. In the generated report, the equation is in line with the text of a paragraph. See “Display Equation in Line with the Text of a Paragraph” on page 12-802. Use this option to fill inline holes in a template.

---

**Note** By default, the bottom of an inline image is aligned with the base line of the surrounding text. If an inline equation image is taller than the surrounding text, you can use the `mlreportgen.dom.VerticalAlign` format to align the image relative to the text base line so that the equation base line matches the text base line. You must experiment to determine the required amount of vertical adjustment.

---

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**SnapshotFormat — Snapshot image format**

`"svg"` (default) | `"png"` | `"emf"`

Snapshot image format, specified as a character vector or string scalar. Supported formats are:

- `"png"` — PNG image.
- `"emf"` — Enhanced metafile. This format is supported only in DOCX output on Windows platforms.
- `"svg"` — Scalable Vector Graphics (SVG).

If the `UseDirectRenderer` property is `true`, only the PNG image format is supported. If your application requires another format, set the `UseDirectRenderer` property to `false`.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**UseDirectRenderer — Whether to use direct equation rendering**

`false` (default) | `true`

Whether to use the direct equation rendering, specified as `true` or `false`. If the value is `true`, the reporter renders an equation directly by using the rendering used by the Live Editor and Simulink Editor. If the value is `false`, the reporter uses a figure window to render an equation.

Direct rendering provides better support for equation markup than the support provided by a figure window. With direct rendering, rendered equations have the same appearance as equations in the Live Editor and Simulink annotations.

Direct rendering does not support:

- Image formats other than PNG.
- Background colors for equations.
- The short name for a color, such as "r". You must use a valid CSS color name, such as "red", or the equivalent RGB triplet or hexadecimal color code. See <https://www.w3.org/wiki/CSS/Properties/color/keywords>.

If your application requires formats other than PNG or a different background color, set the `UseDirectRenderer` property to `false`.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**TemplateSrc — Source of template for this reporter**

[ ] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template specified by the `TemplateSrc` property of this reporter.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**LinkTarget — Hyperlink target for this reporter**

[ ] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.



**Attributes:**

GetAccess	public
SetAccess	public

**Methods****Public Methods**

mlreportgen.report.Equation.createTemplate	Create equation template
mlreportgen.report.Equation.customizeReporter	Create custom equation reporter class
mlreportgen.report.Equation.getClassFolder	Equation class definition file location
copy	Create copy of reporter object and make deep copies of property values that reference a reporter, ReporterLayout, or DOM object
getContentReporter	Get equation content hole reporter
getSnapshotImage	Create equation image and return file path or data URL
getImpl	Get implementation of reporter. If the DisplayInline property is set to true, getImpl(eq, report) returns the snapshot image of the specified equation in the specified report so that you can add the image to a paragraph. Otherwise, getImpl(eq, report) returns a document part.

**Examples****Add an Equation to a Chapter**

Create a report that includes an equation in a chapter. By default, the reporter adds an image of the equation on a separate line of the report.

```
% Import the API packages
import mlreportgen.report.*

% Create the report and chapter
% Add an equation as a separate line in the chapter
rpt = Report('equation', 'docx');
ch = Chapter(Title='Equation');
add(ch,Equation('\int_{0}^{2} x^2\sin(x) dx'));
add(rpt,ch);

% Close and view the report.
close(rpt);
rptview(rpt);
```

### Set Equation Font Size and Color

Create a report that includes an equation on a yellow background in 14-pt font.

```
% Import the API packages
import mlreportgen.report.*

% Create the report and chapter
% Create an Equation reporter and set the FontSize and Color properties
rpt = Report('equation','docx');
ch = Chapter(Title='Equation');
eq = Equation;
eq.Content = '\int_{0}^{2} x^2\sin(x) dx';
eq.FontSize = 14;
eq.Color = 'blue';
eq.BackgroundColor = 'y';
add(ch,eq);
add(rpt,ch);

% Close and view the report
close(rpt);
rptview(rpt);
```

### Display Equation in Line with the Text of a Paragraph

Create an equation that is in line with the text of a paragraph by setting the `DisplayInline` property to `true`. Then, call the `getImpl` method to get the image snapshot of the equation. Add the snapshot of the image to the paragraph.

```
% Import the API packages
import mlreportgen.report.*
import mlreportgen.dom.*

% Create report
% Add equation in line with text in a paragraph of the report
rpt = Report("equation","docx");
eq = Equation("\int_{0}^{2} x^2\sin(x) dx");
eq.DisplayInline = true;
img = getImpl(eq,rpt);
img.Style = {VerticalAlign("-5pt")};
p = Paragraph("Here is an inline equation: ");
p.WhiteSpace = 'preserve';
append(p,img);
append(p," More text ");

add(rpt,p);

% Close and view the report
close(rpt);
rptview(rpt);
```

## Compatibility Considerations

### Word reports with SVG images

*Behavior changed in R2019b*

Starting in R2019b, Scalable Vector Graphics (SVG) images are supported for Microsoft Word reports. For all report types (HTML, PDF, and Word), the default format for equation images is SVG. Word reports that contain SVG images require Word 2016 or a later version. In MATLAB R2019b or a later release, to generate a report with equation images that are compatible with earlier versions of Word, set the `SnapshotFormat` property to a value other than "svg". To specify the equation image format used by default for Word reports in earlier releases of MATLAB, set `SnapshotFormat` to:

- "emf" for a Windows platform
- "png" for a UNIX or Mac platform

## See Also

Text | `mlreportgen.report.Reporter` | `mlreportgen.report.Report`

## Topics

"What Is a Reporter?" on page 1-3

"Fill the Blanks in a Report Form" on page 13-25

"Create an Inline Equation in a Report" on page 17-80

## Introduced in R2017b

## mlreportgen.report.Figure class

**Package:** mlreportgen.report

**Superclasses:** mlreportgen.report.Reporter

Figure reporter

### Description

Create a figure reporter with a title, figure, and caption.

The mlreportgen.report.Figure class is a handle class.

### Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`fig = mlreportgen.report.Figure()` creates a reporter that makes a snapshot of the figure currently open in MATLAB and adds it to a report. Use the figure properties to add a caption or change the figure size. The snapshot image is stored in the temporary folder of the report. When the report is closed, the snapshot image is copied into the report and the image is deleted from the temporary folder. To prevent the snapshot image files from being deleted, use the `Debug` property of the report. See `mlreportgen.report.Report`.

---

**Note** The figure must remain open until the Figure reporter is added to a report.

---

`fig = mlreportgen.report.Figure(source)` creates a reporter that adds the figure specified by `source` and sets the `Source` property to `source`.

`fig = mlreportgen.report.Figure(Name=Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order.

## Properties

### Snapshot — Figure image

mlreportgen.report.FormalImage object

Figure image, specified as an object of the `mlreportgen.report.FormalImage` reporter class. The reporter uses `gcf` to obtain the current MATLAB figure. It uses the formal image reporter to insert the figure into a report. To specify the size of the snapshot or the caption, use the properties of the `FormalImage` object.

---

**Note** The figure reporter initializes the `Snapshot` property. Do not reset this property.

---

### Source — Figure source

character vector | string scalar | graphics handle

Figure source, specified as a:

- Character vector or string scalar that indicates the path to a valid figure file
- Valid graphics handle

### SnapshotFormat — Snapshot image format

'svg' (default) | 'bmp' | 'jpg' | 'png' | 'emf' | 'tif' | 'pdf'

Snapshot image format, specified as a character vector or string scalar. Supported formats are:

- 'svg' — Scalable Vector Graphics.
- 'bmp' — Bitmap image.
- 'jpg' — JPEG image.
- 'png' — PNG image.
- 'emf' — Enhanced metafile, supported only in DOCX output on Windows platforms.
- 'tif' — Tag Image File format, not supported in HTML output.
- 'pdf' — PDF image (supported only in PDF output).

See “Compatibility Considerations” on page 12-811.

### Scaling — Scaling options for figure snapshot image

'auto' (default) | 'custom' | 'none'

Scaling options for the figure snapshot image, specified as a character vector or string scalar.

`Scaling` controls the size of the figure snapshot image in the image file. Supported scaling options are:

- 'auto' — For PDF or Word (DOCX) output, scales the figure snapshot image to fit the current page layout while maintaining its aspect ratio. First, the figure snapshot image is scaled to the page width. If the image height exceeds the page height, the image is scaled down again. This additional scaling ensures that the image fits the current page with an extra one inch spacing. Scaling does not apply to HTML output.
- 'custom' — Scales the figure snapshot image based on the values of the `Height` and `Width` properties.

When you set `Scaling` to `custom` and have large values for the `Height` and `Width` properties, a `java.lang.OutOfMemoryError` can occur during PDF generation. To avoid this error and ensure that the figure fits on the page, use smaller `Height` and `Width` values.

- 'none' — No sizing is performed

---

**Note** The 'auto' and 'custom' options use the MATLAB `print` command to resize the figure. If the figure is too large to fit legibly in the specified space, the `print` command crops the snapshot image. To avoid cropping, you can specify 'none' as the value of the `Scaling` option and use the reporter specified by the `Snapshot` property to size the figure image. This reporter reduces the size

of the text with the rest of the image and a user might need to zoom the image in a viewer to discern fine detail. See “Resize Figure Snapshot Image” on page 12-808.

---

**Height — Height of snapshot image**

character vector | string scalar

Height of snapshot image, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '2in' specifies two inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

Example: '2in'

**Width — Width of snapshot image**

character vector | string scalar

Width of snapshot image, specified as a character vector or string scalar that consists of a number followed by an abbreviation for a unit of measurement. For example, '2in' specifies two inches. Valid abbreviations are:

- px — pixels (default)
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points

Example: '3in'

**PreserveBackgroundColor — Preserve figure background color**

false (default) | true

Preserve the figure background color in the snapshot, specified as `true` or `false`. If `PreserveBackgroundColor` is `true`, the background color of the snapshot is the same as the background color of the figure. If `PreserveBackgroundColor` is `false`, the background color of the snapshot is white.

**TemplateSrc — Source of template for this reporter**

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter

- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

### TemplateName — Name of template for this reporter

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

### LinkTarget — Hyperlink target for this reporter

[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

## Methods

### Public Methods

<code>getSnapshotImage</code>	Get snapshot image path
<code>mlreportgen.report.Figure.getClassFolder</code>	Figure class definition file location
<code>mlreportgen.report.Figure.createTemplate</code>	Create figure template
<code>mlreportgen.report.Figure.customizeReporter</code>	Create custom figure reporter class
<code>copy</code>	Create copy of reporter object and make deep copies of certain property values
<code>getImpl</code>	Get implementation of reporter

## Examples

### Add a Figure to a Report

Add a figure of a surface plot to a report and set the figure caption and height.

```
import mlreportgen.report.*
surf(peaks);
rpt = Report('peaks');
chapter = Chapter();
chapter.Title = 'Figure Example';
add(rpt,chapter);

fig = Figure();
fig.Snapshot.Caption = '3-D shaded surface plot';
fig.Snapshot.Height = '5in';
```

```
add(rpt,fig);
delete(gcf);
rptview(rpt);
```

### Add Multiple Figures to a Report Page

Add two figures to a report. To place them next to each other on the page, use a DOM Table object.

```
import mlreportgen.report.*
import mlreportgen.dom.*
rpt = Report('peaks');

surf(peaks(20));
figure = Figure();
peaks20 = Image(getSnapshotImage(figure,rpt));
peaks20.Width = '3in';
peaks20.Height = [];
delete(gcf);

surf(peaks(40));
figure = Figure();
peaks40 = Image(getSnapshotImage(figure,rpt));
peaks40.Width = '3in';
peaks40.Height = [];
delete(gcf);

t = Table({peaks20,peaks40; 'peaks(20)', 'peaks(40)'});
add(rpt,t);
close(rpt);
rptview(rpt);
```

### Resize Figure Snapshot Image

This example generates a PDF report that illustrates the difference between resizing a figure snapshot image using the `print` command and resizing using the reporter specified by the `Snapshot` property of the `Figure` reporter.

Create a wide MATLAB® figure. Create three `mlreportgen.report.Figure` reporters from the figure and add them to a report.

- The first `Figure` reporter does not resize the figure.
- The second `Figure` reporter uses the `print` command to resize the figure.
- The third `Figure` reporter uses the `Snapshot` reporter to resize the figure.

```
import mlreportgen.report.*

fig = figure();
ax = axes(fig);
plot(ax, rand(1,100));
```



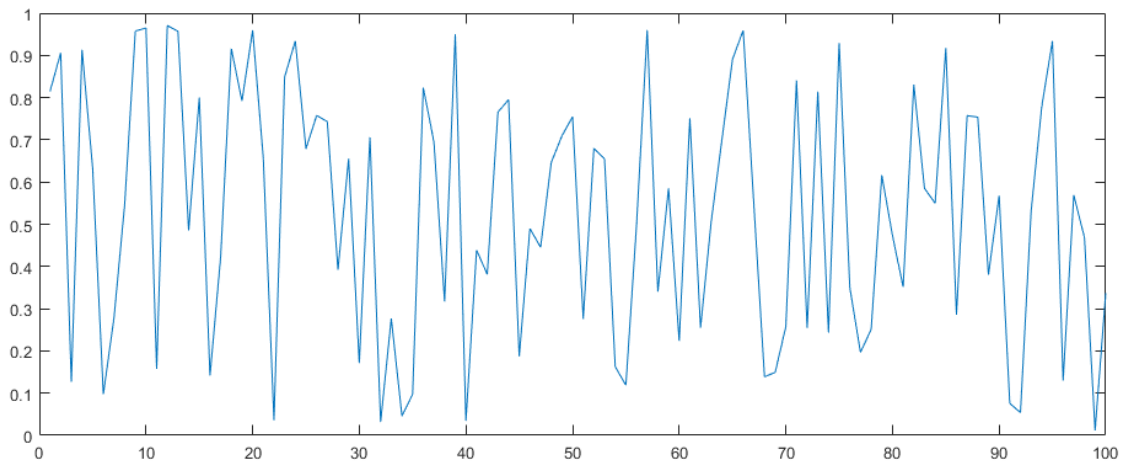
```
pos = fig.Position;
fig.Position = [pos(1) pos(2) 2*pos(3) pos(4)];

rpt = Report('example','pdf');

add(rpt, "Intrinsic figure size");
figReporter0 = Figure(fig);
figReporter0.Scaling = 'none';
add(rpt,figReporter0);

add(rpt, "Resized by print command");
figReporter1 = Figure(fig);
add(rpt,figReporter1);

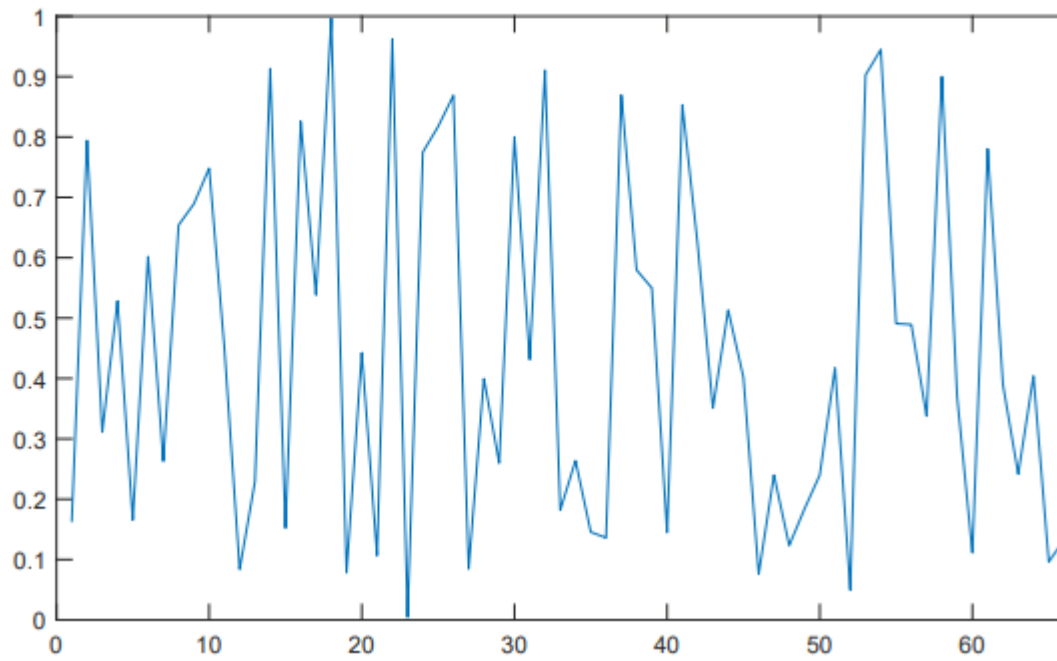
add(rpt, "Resized by snapshot reporter");
figReporter2 = Figure(fig);
figReporter2.Scaling = 'none';
figReporter2.Snapshot.ScaleToFit = true;
add(rpt,figReporter2);
```



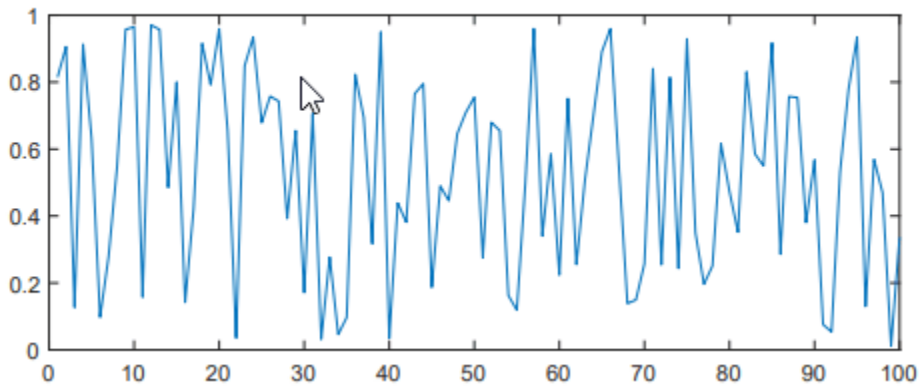
```
close(rpt);
delete(fig)
rptview(rpt);
```

Here are the figures in the generated report:

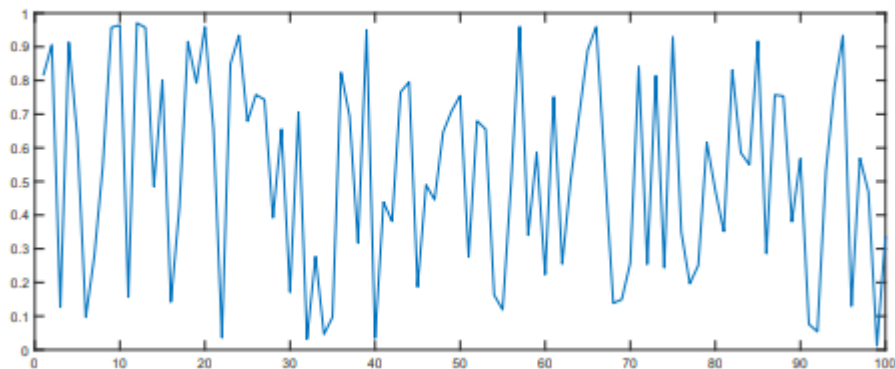
### Intrinsic figure size



### Resized by print command



### Resized by snapshot reporter



## Compatibility Considerations

### Default value of SnapshotFormat is 'svg' for all report types

*Behavior changed in R2019b*

Starting in R2019b, Scalable Vector Graphics (SVG) images are supported for Word reports. For all report types (HTML, PDF, and Word), the default value of the `SnapshotFormat` property is 'svg' and a value of 'auto' indicates 'svg'. In previous releases, the default value of the `SnapshotFormat` property was 'auto', which indicated 'svg' for HTML and PDF reports and 'emf' or 'png' for Word reports, depending on the platform.

Word reports that contain SVG images require Word 2016 or a later version. In MATLAB R2019b or a later release, to generate a report with images that are compatible with earlier versions of Word, set the `SnapshotFormat` property to a value other than 'svg'. To specify the image format used by default in earlier releases of MATLAB, set `SnapshotFormat` to:

- 'emf' for a Windows platform
- 'png' for a UNIX or Mac platform

## See Also

`mlreportgen.report.FormatImage` | `mlreportgen.dom.Table` | `gca` | `mlreportgen.report.Reporter` | `mlreportgen.report.Report`

## Topics

"What Is a Reporter?" on page 1-3

## Introduced in R2017b

## mlreportgen.report.FormalImage class

**Package:** mlreportgen.report

Captioned image reporter

### Description

Create a reporter for an image with a caption.

### Construction

`image = FormalImage()` creates an empty image reporter. Use the reporter properties to set the image source, caption, height, width, and so on. The reporter uses a template to format and number the caption and position it relative to the image. To customize the format, you can specify a custom template or override the template programmatically, using the properties of this reporter.

`image = FormalImage(source)` creates an image reporter that adds the image specified by the source to a report. See the `Image` property.

`image = FormalImage(Name=Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order.

### Input Arguments

#### source — Source of image

string | character array | DOM Image object

Source of image to add to report, specified as a string or character array, or as a DOM Image object. See the `Image` property.

### Properties

#### Image — Source of image

[] | string | character array | DOM Image object

Source of image to add to report, specified as a string or character array, or as a DOM Image object. If you use a string or character array, specify the system path to the image file.

Supported image formats are:

- `.bmp` — Bitmap image
- `.gif` — Graphics Interchange Format
- `.jpg` — JPEG image
- `.png` — PNG image
- `.emf` — Enhanced metafile, supported only in DOCX output on Windows platforms
- `.svg` - Scalable Vector Graphics
- `.tif` - Tag Image File format, not supported in HTML output

- `.pdf` - PDF image (supported only in PDF output)

This reporter inserts the specified image in a paragraph whose style is specified by the template of the reporter. The paragraph style determines the alignment and spacing of the image relative to its caption. To customize the alignment and spacing, customize the `FormalImage` template in the template library for the reporter.

### Caption — Caption of this formal image

[ ] | string | character array | ...

Caption of this formal image, specified as one of these values:

- String or character array
- DOM object
- 1-by-*N* or *N*-by-1 array of strings or DOM objects
- 1-by-*N* or *N*-by-1 cell array of strings, character arrays, and/or DOM objects
- Hole reporter returned by the `getCaptionReporter` method

The caption is numbered automatically and positioned under the image.

Inline content is content that a paragraph can contain. If the caption value is inline content, the reporter uses a template stored in its template library to format the caption. The template automatically numbers the caption using a format that depends on whether the image is in a numbered or unnumbered chapter.

- An image in a numbered chapter has a caption text prefix of the form 'Figure *N.M.*' where *N* is the number of the chapter and *M* is the number of the figure in the chapter. For example, the prefix for the third image in the second chapter of the report is Figure 2.3.
- An image in an unnumbered chapter has a caption text prefix of the form 'Figure *N.*' where *N* is 1 for the first image in the report, 2 for the second image, and so on.

In many non-English locales, the caption prefix is translated to the language and format of the locale. See the `Locale` property of `mlreportgen.report.Report` for a list of translated locales.

### Width — Width of this image

[ ] | string | character array

Width of this image, specified as a string or character array. This property applies only to a formal image whose source you specify as an image path.

The `Width` format is *valueUnits*, where *Units* is an abbreviation for the width units and *value* is the number of units. The table shows the valid *Units* abbreviations.

Units	Units Abbreviation
pixels	px
centimeters	cm
inches	in
millimeters	mm
picas	pc
points	pt

Units	Units Abbreviation
percent	%

If you set the image width, but not the height, the height is scaled to preserve the aspect ratio of the image.

Example: 5in

### Height — Height of this image

[] | string | character array

Height of this image, specified as a string or character array. This property applies only to a formal image whose source you specify as an image path.

The `Height` format is *valueUnits*, where *Units* is an abbreviation for the height units and *value* is the number of units. See the `Width` property for a list of valid *Units* abbreviations.

If you set the image height, but not the width, the width is scaled to preserve the aspect ratio of the image.

### ScaleToFit — Scale image

true (default) | false

Whether to scale this formal image, specified as a logical value. This property specifies whether to scale the image to fit between the margins of a Microsoft Word or PDF page or a table entry.

### Map — Map of hyperlink areas

[] | `mreportgen.dom.ImageMap` object

Map of hyperlink areas in this formal image, specified as an `mreportgen.dom.ImageMap` object. This property applies only to HTML and PDF reports. Use `mreportgen.dom.ImageArea` to define the image areas and then, add them to the map. Image areas are areas in the image that contain hyperlinks to open content in a browser or navigate to another location on the same page.

### TemplateSrc — Source of template for this reporter

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

### TemplateName — Name of template for this reporter

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (TemplateSrc) for this reporter.

### LinkTarget — Hyperlink target for this reporter

[] (default) | character vector | string scalar | mlreportgen.dom.LinkTarget object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an mlreportgen.dom.LinkTarget object. A character vector or string scalar value is converted to a LinkTarget object. The link target immediately precedes the content of this reporter in the output report.

## Methods

createTemplate	Create formal image template
customizeReporter	Create custom formal image reporter class
getCaptionReporter	Get image caption reporter
getClassFolder	Formal image class definition file location
getImageReporter	Get formal image reporter

## Inherited Methods

copy	Create copy of reporter object and make deep copies of certain property values
getImpl	Get implementation of reporter

## Examples

### Add an Image to a Report

Add an empty image reporter to a report and then, set its source, caption, and height.

```
import mlreportgen.report.*
rpt = mlreportgen.report.Report('output','pdf');
chapter = mlreportgen.report.Chapter();
chapter.Title = 'Formal Image Reporter Example';

image = mlreportgen.report.FormalImage();
image.Image = which('ngc6543a.jpg');
image.Caption = 'Cat's Eye Nebula or NGC 6543';
image.Height = '5in';

add(chapter,image);
add(rpt,chapter);
rptview(rpt);
```

### Change Image Caption Color

Add an image to a report. Use default formatting, but change the text color of the caption to red.

```
import mlreportgen.report.*
import mlreportgen.dom.*
```

```

rpt = Report('output', 'pdf');
chapter = Chapter();
chapter.Title = 'Formal Image Reporter Example';

image = FormalImage();
image.Image = which('ngc6543a.jpg');
text = Text('Cat's Eye Nebula or NGC 6543');
text.Color = 'red';
image.Caption = text;

add(chapter, image);
add(rpt, chapter);
rptview(rpt);

```

### Change Image and Caption Formatting

Add an image to a report and override its alignment, caption font, and margins

```

import mlreportgen.report.*
import mlreportgen.dom.*

rpt = Report('output', 'pdf');
chapter = Chapter();
chapter.Title = 'Formal Image Reporter Example';

image = FormalImage();
image.Image = which('ngc6543a.jpg');
image.Height = '5in';

para = Paragraph('System Design Description');
para.Style = {HAlign('left'), FontFamily('Arial'), ...
    FontSize('12pt'), Color('white'), ...
    BackgroundColor('blue'), ...
    OuterMargin('0in', '0in', '.5in', '1in')};
image.Caption = para;

add(chapter, image);
add(rpt, chapter);
rptview(rpt);

```

### Create an Image Map

Create an image map with a defined image area in the upper left and add that image to the report. If you click in the image area, it displays the web page associated with that area.

```

import mlreportgen.report.*;
rpt = Report('test', 'pdf');

image = FormalImage(which('ngc6543a.jpg'));
area = mlreportgen.dom.ImageArea('https://www.google.com', ...
    'Google', 0, 0, 100, 100);
map = mlreportgen.dom.ImageMap;
append(map, area);
image.Map = map;

add(rpt, image);
close(rpt);
rptview(rpt);

```



## See Also

[mlreportgen.report.Reporter](#) | [mlreportgen.report.Report](#) | [mlreportgen.dom.Image](#) | [mlreportgen.dom.ImageArea](#) | [mlreportgen.dom.ImageMap](#) | [mlreportgen.dom.LinkTarget](#)

## Topics

“What Is a Reporter?” on page 1-3

## Introduced in R2017b

## mlreportgen.report.HTMLModuleTabs class

**Package:** mlreportgen.report

**Superclasses:** mlreportgen.report.Reporter

Create tabbed panels

### Description

An mlreportgen.report.HTMLModuleTabs reporter adds a widget consisting of a stack of tabbed panels (module tabs) to an HTML report. Selecting a tab displays the contents of the panel. Use this reporter to display related information in compact form.

---

**Note** Use HTMLModuleTabs reporters only with HTML or single-file HTML reports.

---

The mlreportgen.report.HTMLModuleTabs class is a handle class.

### Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

tabsObj = mlreportgen.report.HTMLModuleTabs() creates an empty HTMLModuleTabs reporter. You must specify the tab labels and content using the TabsData property. Adding an empty HTMLModuleTabs reporter to a report produces an error.

tabsObj = mlreportgen.report.HTMLModuleTabs(Name=Value) sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order.

## Properties

### TabsData — Tab label and panel content

structure array

Tab label and panel content, specified as an array of structures with these fields:

- Label — Tab label, specified as a character vector, a string scalar, or an mlreportgen.dom.Text object. Use a unique label for each tab.
- Content — Panel content, specified as one of these values:
  - A character vector or string scalar
  - A DOM object

- A Report API reporter object

---

**Note** To include multiple DOM objects on one tab, set the Content field to an `mlreportgen.dom.Group` object that contains the DOM objects.

---

### TemplateSrc — Source of template for this reporter

character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

For an HTML report, the type of the template must be `html`. For a single-file report, the type must be `html`.

### TemplateName — Name of template for this reporter

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

### LinkTarget — Hyperlink target for this reporter

[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

## Methods

### Public Methods

<code>mlreportgen.report.HTMLModuleTabs.createTemplate</code>	Create a copy of the HTML module tabs reporter template
<code>mlreportgen.report.HTMLModuleTabs.customizeReporter</code>	Create a custom HTML module tabs reporter
<code>mlreportgen.report.HTMLModuleTabs.getClassFolder</code>	HTML module tabs reporter class definition file location
<code>copy</code>	Create copy of reporter object and make deep copies of certain property values
<code>getImpl</code>	Get implementation of reporter

## Examples

### Report Simulink Model Diagrams as a Tabbed Image Gallery

This example generates a single-file HTML report that has a separate tabbed panel for each system diagram of a Simulink model.

This example requires Simulink and Simulink Report Generator.

Create the report and a chapter.

```
rpt = slreportgen.report.Report("MyReport","html-file");  
open(rpt);
```

```
ch = mlreportgen.report.Chapter("slrgex_sf_car System Diagrams Tabbed Image Gallery");
```

Load the model and find all the diagrams in the model.

```
model_name = "slrgex_sf_car";  
load_system(model_name);
```

```
finder = slreportgen.finder.DiagramFinder(model_name);  
results = find(finder);
```

Create an `mlreportgen.report.HTMLModuleTabs` reporter to contain tabs that correspond to the diagrams. Specify the tab labels and content for each system diagram. The tab label is the system name. The tab content is the system diagram snapshot.

```
moduleTabs = mlreportgen.report.HTMLModuleTabs();  
for result = results  
    moduleTabs.TabsData(end+1).Label = result.Name;  
  
    diag = result.getReporter();  
    moduleTabs.TabsData(end).Content = mlreportgen.dom.Image(diag.getSnapshotImage(rpt));  
end
```

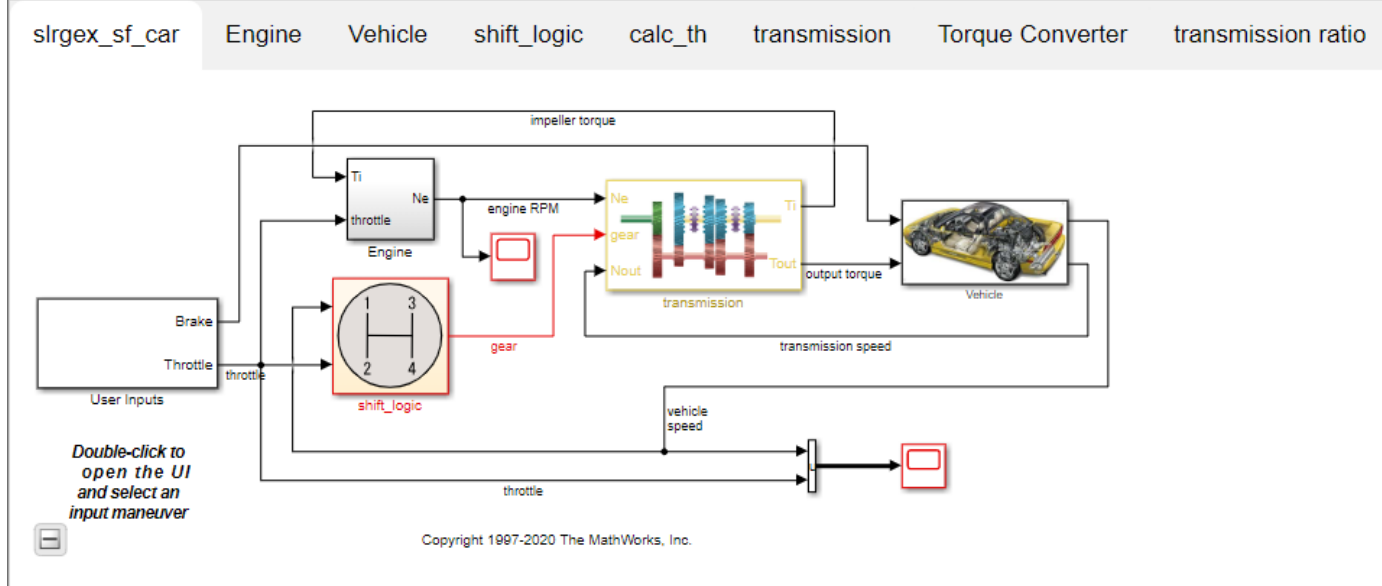
Add the `HTMLModuleTabs` reporter to the chapter and add the chapter to the report. Close and view the report.

```
add(ch,moduleTabs);  
add(rpt,ch);
```

```
close(rpt);  
rptview(rpt);
```

The report opens with the content of the first tab visible. The first tab contains the top-level diagram of the model. To see a different diagram, click the corresponding tab.

## Chapter 1. slrgex\_sf\_car System Diagrams Tabbed Image Gallery



### Create Tabbed Panels with Different Types of Content

This example generates tabbed panels where each panel contains a different type of content. The example also shows how to include multiple DOM objects in the content of a tabbed panel by grouping the DOM objects in an `mlreportgen.dom.Group` object.

Create a report and a chapter.

```
rpt = mlreportgen.report.Report("MyReport", "html");
open(rpt);
ch = mlreportgen.report.Chapter("Tabs with Different Types of Content");
```

Create an `HTMLModuleTabs` reporter and specify the label and content for each tabbed panel. For the last panel, create a `Group` object that contains a paragraph and a table.

```
% Create group from a paragraph and a table
p = mlreportgen.dom.Paragraph('This is a table:');
t = mlreportgen.dom.Table(magic(2));
grp = mlreportgen.dom.Group;
append(grp,p);
append(grp,t);

% Create cell arrays for the labels and content
labels = {'Text','Paragraph','Link','List','Image','Group'};
content = {"This tab contains text as a string.",...
    mlreportgen.dom.Paragraph('This tab contains content using a DOM Paragraph.'),...
    mlreportgen.dom.ExternalLink('http://www.mathworks.com/', "MathWorks"),...
    mlreportgen.dom.UnorderedList(["Coffee", "Tea", "Milk"]),...
    mlreportgen.dom.Image(which("ngc6543a.jpg")),...
    grp};

% Create a structure from the labels and content
tabsdata = struct('Label', labels, 'Content', content);

% Create the HTMLModuleTabs reporter
modTabsObj = mlreportgen.report.HTMLModuleTabs('TabsData', tabsdata);
```

Add the HTMLModuleTabs reporter to the report. Close and view the report.

```
add(ch,modTabsObj);
add(rpt,ch);
close(rpt);
rptview(rpt);
```

The report opens with the content of the first tab visible.

### Chapter 1. Tabs with Different Types of Content

Text Paragraph Link List Image Group

This tab contains text as a string.

Click the Group tab to see that it contains a paragraph and a table.

### Chapter 1. Tabs with Different Types of Content

Text Paragraph Link List Image Group

This is a table:

1	3
4	2

#### See Also

`m\reportgen.dom.Group`

#### Topics

“What Is a Reporter?” on page 1-3

**Introduced in R2020a**

# mlreportgen.report.InlineContent class

**Package:** mlreportgen.report

Fill inline content hole reporter

## Description

Reporter for templates with an inline content hole. An inline hole is a hole within a document body paragraph. Other reporters use this `InlineContent` reporter to fill inline holes in their templates. Reporters have methods that return instances of this object. Using these object instances, you can customize the format of the content used to fill the holes in their templates. For example, for the `TitlePage` reporter, the `getTitleReporter` method returns the instance that the `TitlePage` reporter uses to fill the `TitlePageTitle` hole in its template. To customize the title format, specify a custom template for the `InlineContent` reporter returned by the `getTitleReporter` method.

---

**Note** Reporters create instances of this object. You do not need to create this object yourself.

---

## Properties

### HoleId — ID of hole

string

ID of hole to be filled by this reporter, specified as a string.

### Content — Content of hole

string | character array | ...

Content of hole to be filled by this reporter, specified as one of these values:

- String or character array
- Inline DOM object
- 1-by-*N* or *N*-by-1 array of strings or DOM objects
- 1-by-*N* or *N*-by-1 cell array of strings, character arrays, and/or DOM objects

### TemplateSrc — Source of template for this reporter

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the

TemplateSrc property is empty, this reporter uses the default reporter template for the output type of the report.

**TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (TemplateSrc) for this reporter.

**LinkTarget — Hyperlink target for this reporter**

[] (default) | character vector | string scalar | mlreportgen.dom.LinkTarget object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an mlreportgen.dom.LinkTarget object. A character vector or string scalar value is converted to a LinkTarget object. The link target immediately precedes the content of this reporter in the output report.

**See Also**

mlreportgen.report.Report | mlreportgen.report.Reporter

**Topics**

“What Is a Reporter?” on page 1-3

**Introduced in R2017b**



# mlreportgen.report.ListOfCaptions class

**Package:** mlreportgen.report

**Superclasses:** mlreportgen.report.Reporter

List of captions reporter

## Description

Use an object of the `mlreportgen.report.ListOfCaptions` class to create a reporter that adds a section for a list of captioned report elements to a report. To create a reporter for a list of figures with captions or tables with titles, use objects of the `mlreportgen.report.ListOfFigure` and `mlreportgen.report.ListOfTables` classes, respectively. To create a reporter for a list for other types of report elements, such as equations, use a `ListOfCaptions` object.

To identify the report elements to include in the list of captions:

- 1 Choose a numbering stream name, for example, `equation`. Set the `AutoNumberStreamName` property of the `ListOfCaptions` object to the numbering stream name.
- 2 Create captions for the report objects as `mlreportgen.dom.Paragraph` objects.
- 3 Associate the `Paragraph` objects with the numbering stream name by using an `mlreportgen.dom.AutoNumber` object.

The generated list of captions contains the captions associated with the specified numbering stream name. The list entries link to the captions in the report. In PDF and Word reports, the list also includes the page numbers of the captions in the report. A leader fills the space between a caption and its page number.

The way the list of captions is generated depends on the report type:

- PDF — The Report API generates the list during report generation.
- Word — The Report API generates a placeholder for the list. To generate the list items, you must update the Word document in your report generation program or in Word. See “Update a Table of Contents or Generated List in a Word Document” on page 3-37.
- HTML — The Report API generates a placeholder for the list. When the report opens in an HTML browser, the browser generates the list in a sidebar.

---

**Note** You can use a `ListOfCaptions` reporter for captions that follow a report element or titles that precede a report element.

---

The `mlreportgen.report.ListOfCaptions` class is a `handle` class.

### Class Attributes

`HandleCompatible` `true`

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`loc = mlreportgen.report.ListOfCaptions()` creates a `ListOfCaptions` reporter with default property values. Specify the list of captions section title by setting the `Title` property.

`lof = mlreportgen.report.ListOfCaptions(title)` creates a `ListOfCaptions` reporter with the `Title` property set to the specified title.

`lof = mlreportgen.report.ListOfCaptions(Name=Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order.

## Properties

### Title — List of captions section title

`[]` (default) | string scalar | character vector | inline DOM object | `mlreportgen.report.Title` object | ...

List of captions section title, specified as one of these values:

- String scalar or character vector
- Inline DOM object, such as an `mlreportgen.dom.Text` object
- 1-by-*N* or *N*-by-1 array of string scalars or inline DOM objects
- 1-by-*N* or *N*-by-1 cell array that contains any combination of strings, character vectors, or inline DOM objects
- `mlreportgen.report.Title` object returned by the `getTitleReporter` method

### AutoNumberStreamName — Name of numbering stream

`' '` (default) | character vector | string scalar

Name of numbering stream, specified as a character vector or string scalar.

### LeaderPattern — Type of leader

`'.'` (default) | `'dots'` | `' '` | `'space'`

Type of leader to use between the caption and the page number, specified as one of these character vectors or string scalars:

- `'.'` or `'dots'`
- `' '` or `'space'`

This property applies only to PDF reports. Word reports always have a dots leader. HTML reports do not have a leader.

### Layout — Page layout

`mlreportgen.report.ReporterLayout` object (default)

Page layout for the list of captions section, specified as an `mlreportgen.report.ReporterLayout` object. Use the properties of the `ReporterLayout` object to override some of the default page layout properties, such as page orientation.

**TemplateSrc — Source of template for this reporter**

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

**TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

**LinkTarget — Hyperlink target for this reporter**[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

**Methods****Public Methods**

<code>mlreportgen.report.ListOfCaptions.createTemplate</code>	Create list of captions reporter template
<code>mlreportgen.report.ListOfCaptions.customizeReporter</code>	Create custom list of captions reporter class
<code>mlreportgen.report.ListOfCaptions.getClassFolder</code>	Get location of list of captions reporter class definition file
<code>copy</code>	Create copy of reporter object and make deep copies of certain property values
<code>getImpl</code>	Get implementation of reporter
<code>getTitleReporter</code>	Get list of captions title reporter

**Examples****Generate a List of Equations Section in a Report**

This example generates a section for a list of the captions of the equations in a report. You can use the same procedure to generate a list of captions for other report elements. The example identifies the captions to include in the list by associating the captions with the automatic numbering stream

that has the name `equation`. You can use any name for the numbering stream as long as the name matches the value of the `AutoNumberStream` property of the `ListOfCaptions` object that represents the list of captions.

Import the DOM and Report API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
import mlreportgen.report.*
```

Create a report.

```
rpt = Report("My Report", "pdf");
```

Add a table of contents to the report.

```
toc = TableOfContents;
append(rpt, toc);
```

Create a list of captions section and add it to the report. Specify that the elements to include in the list are associated with the number stream name `equation`.

```
loc = ListOfCaptions();
loc.Title = "List of Equations";
loc.AutoNumberStreamName = "equation";
append(rpt, loc);
```

Create a chapter and add an equation to the chapter.

```
ch = Chapter("Physics Principles");
eq = Equation("e = m * c^2");
append(ch, eq);
```

Create a paragraph for the equation caption.

```
p = Paragraph("Equation ");
```

Create an automatic numbering stream with the name `equation` and associate it with the paragraph.

```
append(p, AutoNumber("equation"));
```

Increment the counter for the numbering stream.

```
p.Style = {HAlign("center"), CounterInc("equation"), WhiteSpace("preserve")};
```

Append the rest of the caption text to the paragraph and append the paragraph to the chapter. Append the chapter to the report.

```
append(p, ' Mass-energy equivalence');
append(ch, p);
append(rpt, ch);
```

Close and view the report.

```
close(rpt);
rptview(rpt);
```

Here is the list of equations in the report:

## List of Equations

[Equation 1 Mass-energy equivalence](#)..... 1

### See Also

[mlreportgen.report.ListOfFigures](#) | [mlreportgen.report.ListOfTables](#) |  
[mlreportgen.dom.AutoNumber](#) | [mlreportgen.dom.CounterInc](#) |  
[mlreportgen.report.TableOfContents](#) | [mlreportgen.report.ReporterLayout](#)

### Topics

“Create a List of the Captions or Titles of Related Report Elements in a Report API Report” on page 3-46  
“Update a Table of Contents or Generated List in a Word Document” on page 3-37  
“Automatically Number Document Content” on page 13-101

### Introduced in R2020b

## mlreportgen.report.ListOfFigures class

**Package:** mlreportgen.report

**Superclasses:** mlreportgen.report.Reporter

List of figures reporter

### Description

Use an object of the `mlreportgen.report.ListOfFigures` class to create a reporter that adds a list of figures section to a report. The generated list of figures contains the captions of the figures (images) that are represented as `mlreportgen.report.FormalImage` or `mlreportgen.report.Figure` objects in your report generation program. The list entries link to the captions in the report. In PDF and Word reports, the list also includes the page numbers of the captions in the report. A leader fills the space between a caption and its page number.

The way the list of figures is generated depends on the report type:

- PDF — The Report API generates the list during report generation.
- Word — The Report API generates a placeholder for the list. To generate the list items, you must update the Word document in your report generation program or in Word. See “Update a Table of Contents or Generated List in a Word Document” on page 3-37.
- HTML — The Report API generates a placeholder for the list. When the report opens in an HTML browser, the browser generates the list in a sidebar.

The `mlreportgen.report.ListOfFigures` class is a `handle` class.

### Class Attributes

Abstract true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`lof = mlreportgen.report.ListOfFigures()` creates a `ListOfFigures` reporter with default property values.

`lof = mlreportgen.report.ListOfFigures(title)` creates a `ListOfFigures` reporter with the `Title` property set to the specified title.

`lof = mlreportgen.report.ListOfFigures(Name=Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order.

### Properties

#### Title — List of figures section title

[ ] | string scalar | character vector | inline DOM object | mlreportgen.report.Title object | ...

List of figures section title, specified as one of these values:

- String scalar or character vector
- Inline DOM object, such as an `mlreportgen.dom.Text` object
- 1-by-*N* or *N*-by-1 array of string scalars or inline DOM objects
- 1-by-*N* or *N*-by-1 cell array that contains any combination of strings, character vectors, or inline DOM objects
- `mlreportgen.report.Title` object returned by the `getTitleReporter` method

If you do not specify the `Title` property, the title is `List of Figures`.

### **LeaderPattern — Type of leader**

`'.'` (default) | `'dots'` | `' '` | `'space'`

Type of leader to use between the caption and the page number, specified as one of these character vectors or string scalars:

- `'.'` or `'dots'`
- `' '` or `'space'`

This property applies only to PDF reports. Word reports always have a dots leader. HTML reports do not have a leader.

### **Layout — Page layout**

`mlreportgen.report.ReporterLayoutobject` (default)

Page layout for the list of figures section, specified as an `mlreportgen.report.ReporterLayout` object. Use the properties of the `ReporterLayout` object to override the some of the default page layout properties, such as page orientation.

### **TemplateSrc — Source of template for this reporter**

`[]` (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

### **TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

**LinkTarget — Hyperlink target for this reporter**

[ ] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

**Methods****Public Methods**

<code>mlreportgen.report.ListOfFigures.createTemplate</code>	Create list of figures reporter template
<code>mlreportgen.report.ListOfFigures.customizeReporter</code>	Create custom list of figures reporter class
<code>mlreportgen.report.ListOfFigures.getClassFolder</code>	Get location of list of figures reporter class definition file
<code>copy</code>	Create copy of reporter object and make deep copies of certain property values
<code>getImpl</code>	Get implementation of reporter
<code>getTitleReporter</code>	Get list of figures title reporter

**Examples****Generate a List of Figures Section in a Report**

This example uses an `mlreportgen.report.TableOfContents` reporter to add a table of contents to a report and an `mlreportgen.report.ListOfFigures` reporter to add a section for a list of figures. The figures (images) in the report are represented as `mlreportgen.report.Figure` or `mlreportgen.report.FormalImage` objects so that the `ListOfFigures` reporter includes them in the list of figures.

Import the Report API package so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
```

Create a report.

```
rpt = Report("myFigureReport","pdf");
open(rpt);
```

Create a table of contents and add it to the report.

```
toc = TableOfContents();
append(rpt,toc);
```

Create a list of figures section and add it to the report.

```
lof = ListOfFigures();
lof.Title = "My List of Figures";
append(rpt,lof);
```

Create a chapter and add a captioned image and figure to the chapter.

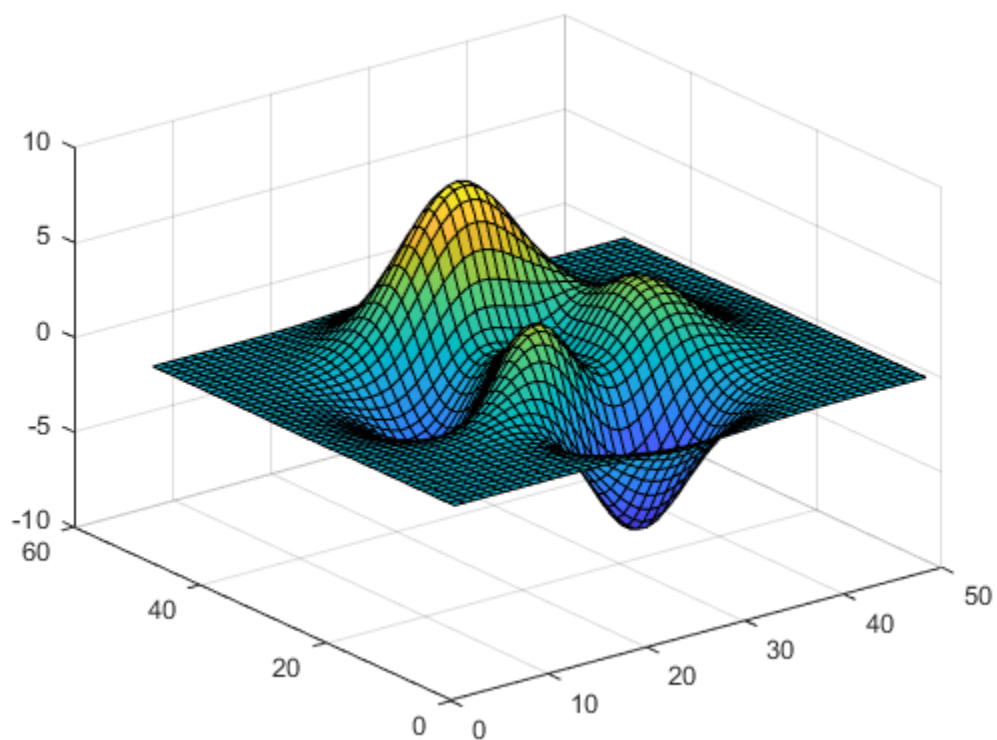
```
ch = Chapter("Images");
image1 = FormalImage(which("peppers.png"));
```

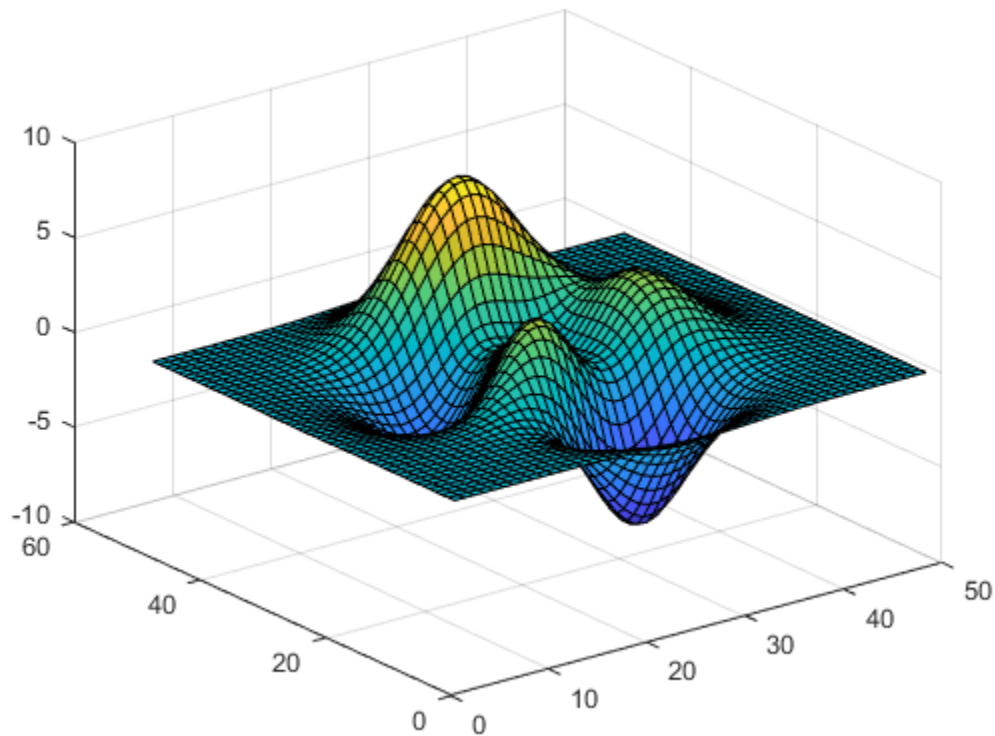


```
image1.Caption = "Peppers";  
image1.Height = "5in";  
append(ch,image1);  
surf(peaks);  
fig = Figure();  
fig.Snapshot.Caption = '3-D shaded surface plot';  
fig.Snapshot.Height = '5in';  
append(ch,fig);
```

Add the chapter to the report. Close and view the report.

```
append(rpt,ch);
```





```
delete(gcf);
close(rpt);
rptview(rpt);
```

Here is the list of figures in the report:

### My List of Figures

<a href="#">Figure 1.1. Peppers</a> .....	1
<a href="#">Figure 1.2. 3-D shaded surface plot</a> .....	2

### See Also

`mlreportgen.report.ListOfCaptions` | `mlreportgen.report.ListOfTables` |  
`mlreportgen.report.TableOfContents` | `mlreportgen.report.ReporterLayout` |  
`mlreportgen.report.FormalImage` | `mlreportgen.report.Figure`

### Topics

“Create a List of Figures or Tables in a Report API Report” on page 3-40  
 “Update a Table of Contents or Generated List in a Word Document” on page 3-37

**Introduced in R2020b**

# mlreportgen.report.ListOfTables class

**Package:** mlreportgen.report

**Superclasses:** mlreportgen.report.Reporter

List of tables reporter

## Description

Use an object of the `mlreportgen.report.ListOfTables` class to create a reporter that adds a list of tables section to a report. The generated list of tables contains the titles of the tables that are represented as `mlreportgen.report.BaseTable` objects in your report generation program. The list items link to the titles in the report. In PDF and Word reports, the list also includes the page numbers of the titles in the report. A leader fills the space between a title and the page number.

The way the list of tables is generated depends on the report type:

- PDF — The Report API generates the list during report generation.
- Word — The Report API generates a placeholder for the list. To generate the list items, you must update the Word document in your report generation program or in Word. See “Update a Table of Contents or Generated List in a Word Document” on page 3-37.
- HTML — The Report API generates a placeholder for the list. When the report opens in an HTML browser, the browser generates the list in a sidebar.

The `mlreportgen.report.ListOfTables` class is a `handle` class.

## Class Attributes

`HandleCompatible` true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`lot = mlreportgen.report.ListOfTables()` creates a `ListOfTables` reporter with default property values.

`lot = mlreportgen.report.ListOfTables(title)` creates a `ListOfTables` reporter with the `Title` property set to the specified title.

`lot = mlreportgen.report.ListOfTables(Name=Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order.

## Properties

### Title — List of tables section title

[ ] | string scalar | character vector | inline DOM object | mlreportgen.report.Title object | ...

List of tables section title, specified as one of these values:

- String scalar or character vector
- Inline DOM object, such as an `mlreportgen.dom.Text` object
- 1-by-*N* or *N*-by-1 array of string scalars or inline DOM objects
- 1-by-*N* or *N*-by-1 cell array that contains any combination of strings, character vectors, or inline DOM objects
- `mlreportgen.report.Title` object returned by the `getTitleReporter` method

If you do not specify the `Title` property, the title is `List of Tables` in the generated list of tables.

### **LeaderPattern — Type of leader**

`'.'` (default) | `'dots'` | `' '` | `'space'`

Type of leader to use between the title and the page number, specified as one of these character vectors or string scalars:

- `'.'` or `'dots'`
- `' '` or `'space'`

This property applies only to PDF reports. Word reports always have a dots leader. HTML reports do not have a leader.

### **Layout — Page layout**

`mlreportgen.report.ReporterLayoutobject` (default)

Page layout for the list of tables section, specified as an `mlreportgen.report.ReporterLayout` object. Use the properties of the `ReporterLayout` object to override the some of the default page layout properties, such as page orientation.

### **TemplateSrc — Source of template for this reporter**

`[]` (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

### **TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

**LinkTarget — Hyperlink target for this reporter**

[] (default) | character vector | string scalar | mlreportgen.dom.LinkTarget object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an mlreportgen.dom.LinkTarget object. A character vector or string scalar value is converted to a LinkTarget object. The link target immediately precedes the content of this reporter in the output report.

**Methods****Public Methods**

mlreportgen.report.ListOfTables.createTemplate	Create list of tables reporter template
mlreportgen.report.ListOfTables.customizeReporter	Create custom list of tables reporter class
mlreportgen.report.ListOfTables.getClassFolder	Get location of list of tables reporter class definition file
copy	Create copy of reporter object and make deep copies of certain property values
getImpl	Get implementation of reporter
getTitleReporter	Get list of tables title reporter

**Examples****Generate a List of Tables Section in a Report**

This example uses an mlreportgen.report.TableOfContents reporter to add a table of contents to a report and an mlreportgen.report.ListOfTables reporter to add a section for a list of tables. The tables in the report are represented as mlreportgen.report.BaseTable objects so that the ListOfTables reporter includes them in the list of tables.

Import the Report API package so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
```

Create a report.

```
rpt = Report("myTableReport", "pdf");
open(rpt);
```

Create a table of contents and add it to the report.

```
toc = TableOfContents();
append(rpt, toc);
```

Create a list of tables section and add it to the report.

```
lot = ListOfTables();
lot.Title = "My List of Tables";
append(rpt, lot);
```

Create a chapter and add captioned tables to the chapter.

```
ch = Chapter("Magic Squares");
t1 = BaseTable(magic(2));
t1.Title = "Order Two Magic Square";
```

```
append(ch,t1);  
t2 = BaseTable(magic(3));  
t2.Title = "Order Three Magic Square";  
append(ch,t2);
```

Add the chapter to the report. Close and view the report.

```
append(rpt,ch);  
close(rpt);  
rptview(rpt);
```

Here is the list of tables in the report:

### **My List of Tables**

<a href="#">Table 1.1. Order Two Magic Square</a> .....	1
<a href="#">Table 1.2. Order Three Magic Square</a> .....	1

### **See Also**

`mlreportgen.report.ListOfCaptions` | `mlreportgen.report.ListOfFigures` |  
`mlreportgen.report.TableOfContents` | `mlreportgen.report.BaseTable` |  
`mlreportgen.report.ReporterLayout`

### **Topics**

“Create a List of Figures or Tables in a Report API Report” on page 3-40  
“Update a Table of Contents or Generated List in a Word Document” on page 3-37

### **Introduced in R2020b**

# mlreportgen.report.MATLABCode class

**Package:** mlreportgen.report

MATLAB code reporter

## Description

Use an object of the mlreportgen.report.MATLABCode class to include syntax-highlighted MATLAB code in a report.

The mlreportgen.report.MATLABCode class is a handle class.

## Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

reporter = mlreportgen.report.MATLABCode() creates an empty MATLABCode reporter object based on the default template. You must specify the MATLAB code file by setting the FileName property, or specify the code content by setting the Content property. Use other properties to specify reporter options.

reporter = mlreportgen.report.MATLABCode(filename) creates a MATLABCode reporter with the FileName property set to filename.

reporter = mlreportgen.report.MATLABCode(Name=Value) sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order.

## Properties

### FileName — Path and file name of MATLAB code file

[ ] (default) | character vector | string scalar

Path and file name of the file that contains MATLAB code, specified as a character vector or string scalar. The file can have a .m or .mlx extension. If you set this property, the MATLABCode reporter sets the Content property to a string scalar that contains the code contained in the specified file.

### Attributes:

GetAccess	public
SetAccess	public
Dependent	true
NonCopyable	true

**Content — MATLAB code**

[] (default) | character vector | string scalar

MATLAB code, specified as a character vector or string scalar. Set this property only if the `FileName` property is not set.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>
<code>Dependent</code>	<code>true</code>
<code>NonCopyable</code>	<code>true</code>

**SmartIndent — Whether to apply smart indenting to the code**

false (default) | true

Whether to apply smart indenting to the code, specified as `true` or `false`.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**IncludeComplexity — Whether to include code complexity**

false (default) | true

Whether to include code complexity, specified as `true` or `false`. If the value is `true`, the report includes the McCabe cyclomatic complexity of each function that the MATLAB code contains.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**ComplexityReporter — Code complexity reporter**`mlreportgen.report.BaseTable`

Code complexity reporter, specified as an `mlreportgen.report.BaseTable` object. The `BaseTable` reporter is used to report and format the code complexity tabular data. The default value of this property is a `BaseTable` object with the `TableStyleName` property set to "MATLABCodeTable" and the other properties set to default values. You can customize the appearance of the table by customizing the default reporter or by replacing it with a custom `BaseTable` reporter. Any content that you specify in the `Title` property of the default or the replacement `BaseTable` reporter appears before the title in the generated report.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**TemplateSrc — Source of template for this reporter**

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter



- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

### TemplateName — Name of template for this reporter

"MATLABCode" (default) | character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

### LinkTarget — Hyperlink target for this reporter

[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

## Methods

### Public Methods

<code>mlreportgen.report.MATLABCode.createTemplate</code>	Create copy of <code>mlreportgen.report.MATLABCode</code> reporter template
<code>mlreportgen.report.MATLABCode.customizeReporter</code>	Create subclass of <code>mlreportgen.report.MATLABCode</code> class
<code>mlreportgen.report.MATLABCode.getClassFolder</code>	Get location of folder that contains <code>mlreportgen.report.MATLABCode</code> class definition file
<code>copy</code>	Create copy of reporter object and make deep copies of certain property values
<code>getImpl</code>	Get implementation of reporter

## Examples

### Include a Syntax-Highlighted MATLAB Function in a Report

Report the syntax-highlighted code for the function `myAdd.m`.

Create a report.

```
import mlreportgen.dom.*
import mlreportgen.report.*

rpt = Report("MyReport", "pdf");
```

Create a chapter.

```
chap = Chapter("The myAdd Function");
```

Create a MATLABCode reporter to report on the content of myAdd.m.

```
mCode = MATLABCode("myAdd.m");
```

Add the reporter to the chapter and the chapter to the report.

```
append(chap,mCode);  
append(rpt, chap);
```

Close the report and open the viewer.

```
close(rpt);  
rptview(rpt);
```

Here is the syntax-highlighted code in the report.

### Chapter 1. The myAdd Function

```
function c = myAdd(a,b)  
%myAdd Add two numbers  
c = a + b;  
end
```

## See Also

[mlreportgen.report.Reporter](#) | [mlreportgen.report.MATLABVariable](#) | [mlreportgen.report.BaseTable](#) | [checkcode](#)

**Introduced in R2021a**

# mlreportgen.report.MATLABVariable class

**Package:** mlreportgen.report

MATLAB variable reporter

## Description

Create a reporter that reports on a MATLAB variable.

## Construction

`rptr = mlreportgen.report.MATLABVariable()` creates a MATLAB variable reporter based on a default template. Before adding this reporter to a report, use its properties to specify the variable name on which to report.

`rptr = mlreportgen.report.MATLABVariable(variable)` creates a MATLAB variable reporter for the specified MATLAB variable. To specify a local variable, specify its name, for example, `MATLABVariable(x)`. To specify a MATLAB workspace variable, specify its name as a string or character array, for example, `MATLABVariable('x')`. To specify other report options, use the properties of this reporter.

`rptr = mlreportgen.report.MATLABVariable(Name=Value)` creates a MATLAB variable reporter with options specified by one or more `Name=Value` pair arguments. `Name` is a property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1=Value1, ..., NameN=ValueN`.

## Input Arguments

### **variable** — MATLAB variable name

string scalar | character vector | local variable name

MATLAB variable name, specified as a string scalar, character vector, or local variable name. To specify a local variable, specify its name, for example, `MATLABVariable(x)`. To specify a MATLAB workspace variable, specify its name as a string scalar or character vector, for example, `MATLABVariable('x')`. For more information on the input, see the `Variable` property.

## Properties

### **Variable** — MATLAB variable name

string scalar | character vector

MATLAB variable name, specified as a string scalar or character vector. The specified variable can be any of these data types:

- Character or character array
- String
- Cell vector or cell array
- Logical scalar, logical vector, or logical array

- Numeric scalar, numeric array, or numeric vector
- MATLAB table
- MATLAB object vector or object array
- Simulink object
- Stateflow object
- Graphics object
- MATLAB structure, structure vector, or structure array
- Enumeration

**Location — Location of variable**

"MATLAB" (default) | "MAT-File" | "Global" | "Local" | "Model" | "User-Defined"

Location of variable, specified as one of these strings or as character arrays:

- "MATLAB" — Base workspace
- "MAT-File" — MAT-file specified in the `FileName` property of this reporter
- "Global" — Global name space
- "Local" — Local name space, typically the name space of the function or workspace in which this reporter was created
- "Model" — Workspace of the Simulink Report Generator model specified in the `FileName` property of this reporter
- "User-Defined" — `setVariableValue` method set the value to report and set this property to "User-Defined"

**FileName — Name of file or model that contains the variable**

string scalar | character vector

Name of MAT-file or Simulink model that contains the variable, specified as a string scalar or as a character vector. This property applies only if the `Location` property value is `MAT-File` or `Model`. If `Location` is `MAT-File`, the file name is the name of the MAT-file from which to obtain the variable. If `Location` is `Model`, the `FileName` is the name of the Simulink model file that contains the variable.

**FormatPolicy — Format of variable values**

"Auto" (default) | "Table" | "Paragraph" | "Inline Text"

Format of the variable values, specified as one of these strings or character vectors:

- "Auto" — Formats the variable values as a table or a paragraph, depending on the data type of the value.

Data types formatted as a table include:

- Cell array
- Logical array
- Numeric array
- MATLAB table
- Simulink object
- Stateflow object

- Graphics object
- MATLAB structure or structure array
- MATLAB object or object array
- MATLAB enumeration class that defines properties

Data types formatted as a paragraph include:

- Cell vector
- Logical scalar or vector
- Numeric scalar or vector
- Character or character array
- String
- MATLAB structure vector
- MATLAB object vector
- MATLAB enumeration class that does not define properties
- "Table" — Formats the variable values in a table. Variables that by default appear as paragraphs are formatted instead as table entries. Variables that are hierarchically structured objects, such as a MATLAB structures, MATLAB objects, Simulink objects, Stateflow objects, or graphics objects, can have properties that are themselves objects. In that case, the hierarchy is flattened and the property value is displayed as a hyperlink to a table of the properties of that object. The object property table also has a hyperlink back to the original table.
- "Paragraph" — Format the variable values as a paragraph. Variables that by default are formatted as tables are flattened and formatted as a paragraph.
- "InLine Text" — Formats the variable in line with the surrounding text.

### **TableReporter — Table reporter for variable values**

`mlreportgen.report.BaseTable`

Table reporter used by this `MATLABVariable` reporter to format variable values, specified as an `mlreportgen.report.BaseTable` object. To customize the appearance of the table, modify the default `BaseTable` reporter properties or replace it with a customized `BaseTable` reporter. If you add content to the `Title` property of the default or customized reporter, that content appears in front of the table title in the generated report.

### **ParagraphFormatter — Paragraph formatter for MATLAB variable**

`mlreportgen.dom.Paragraph` object

Paragraph formatter object to format the value of the MATLAB variable, specified as an `mlreportgen.dom.Paragraph` object. To customize the appearance of the paragraph, modify the `DOM Paragraph` object properties or replace the object with a customized `Paragraph` object. If you add content to the default or replacement paragraph object, that content appears in front of the variable content in the generated report.

### **TextFormatter — Text formatter for MATLAB variable**

`mlreportgen.dom.Text` object

Text formatter object to format MATLAB variable text values in tables or paragraphs, specified as an `mlreportgen.dom.Text` object. To customize the appearance of the text, modify the `DOM Text` object properties or replace the object with a customized `Text` object. If you add content to the

default or replacement text object, that content appears in front of the variable content in the generated report.

**MaxCols — Maximum number of table columns to display**

32 (default) | positive integer

Maximum number of table columns to display, specified as a positive integer. For array variables reported using a table, if the number of columns is greater than the value of the `MaxCols` property, the table is sliced vertically. Slicing divides the table into multiple tables.

**DepthLimit — Maximum number of nested levels to report**

10 (default) | nonnegative integer

Maximum number of levels to report for a variable that is a structured object or an array of structured objects, specified as a nonnegative integer. Levels less than or equal to the value of `DepthLimit` are flattened into a sequence of interlinked tables (see the `FormatPolicy` property). Levels greater than the depth limit are not reported. If you set the `DepthLimit` property to 0, structured objects are not expanded.

**ObjectLimit — Maximum number of nested objects to report**

200 (default) | positive integer

Maximum number of objects in an object hierarchy to report, specified as a positive integer.

**IncludeTitle — Whether to include title**

true (default) | false

Whether to include a title, specified as `true` or `false`. The title contains the variable name and optionally, the data type. If `IncludeTitle` is `true`, the title is included. By default, the title includes only the name of the variable. To include the data type of the variable, set the `ShowDataType` property to `true`.

**Title — Title of variable to report**

[] (default) | character vector | string scalar | `mlreportgen.dom.Text` object | `mlreportgen.dom.InternalLink` object | `mlreportgen.dom.ExternalLink` object

Title of variable to report, specified as a character vector, string scalar, `mlreportgen.dom.Text` object, `mlreportgen.dom.InternalLink` object, or `mlreportgen.dom.ExternalLink` object.

If the `FormatPolicy` property is set to "Inline Text" and the `Title` property is set to:

- A DOM object, the formatting specified by the DOM object is ignored
- An `mlreportgen.dom.InternalLink` or `mlreportgen.dom.ExternalLink` object, the link text is used for the title, but the title is not a link

In both cases, to format the title, use the `TextFormatter` property of this `MATLABVariable` reporter.

If you do not specify the `Title` property, the title is the variable name.

**ShowDataType — Whether to show data type of variable in title**

false (default) | true

Whether to show the data type of the variable in the title, specified as `true` or `false`.

**ShowEmptyValues — Whether to show properties that have empty values**

true (default) | false

Whether to show properties that have empty values, specified as a true or false. The ShowEmptyValues property applies only to MATLAB object, Simulink object, and Stateflow object variables.

**ShowDefaultValues — Whether to show properties that use default values**

true (default) | 0

Whether to show properties that use the default value, specified as true or false. The ShowDefaultValues property applies only to MATLAB object, Simulink object, and Stateflow object variables.

**PropertyFilterFcn — Function or expression to filter properties of a reported variable**

[] (default) | function handle | string scalar | character vector

Function or expression to filter the properties of a variable from a report. Specify a function as a function handle. Specify an expression as a string scalar or character vector. This property applies only to variables that contain objects. If you do not provide PropertyFilterFcn, all properties of the variable are included in the report.

If you provide a function handle, the associated function must:

- Take these arguments:
  - `variableName` — Name of the variable being reported
  - `variableObject` — The variable being reported
  - `propertyName` — Name of the property of the variable being reported
- Return true to filter the specified property from the report, or false to include the property in the report.

For example, this code prevents the display of the NumRegions and NumHoles properties of a polyshape object.

```
import mlreportgen.report.*

rpt = mlreportgen.report.Report('variableRpt','pdf');
open(rpt);

pgon = polyshape([0 0 2 2],[2 0 0 2]);
mlVar = mlreportgen.report.MATLABVariable(pgon);
mlVar.PropertyFilterFcn = @varPropertyFilter;
add(rpt,mlVar);

close(rpt);
rptview(rpt);

function tf = varPropertyFilter(~,variableObject,propertyName)
if isa(variableObject, "polyshape")
    tf = (propertyName == "NumRegions") || ...
        (propertyName == "NumHoles");
end
end
```

If you provide a string scalar or a character vector, it must contain an expression. The expression:

- Can use the variables `variableName`, `variableObject`, and `propertyName`

- Must set the variable `isFiltered` to `true` to filter the specified property from the report, or `false` to include the property in the report

For example, this code filters the `NumHoles` property of `polyshape` object from the report.

```
import mlreportgen.report.*

rpt = mlreportgen.report.Report('variable.rpt','pdf');
open(rpt);

pgon = polyshape([0 0 2 2],[2 0 0 2]);
mlVar = mlreportgen.report.MATLABVariable(pgon);
mlVar.PropertyFilterFcn = "isFiltered = " + ...
    "isa(variableObject, 'polyshape') && " + ...
    "propertyName == 'NumHoles'";
add(rpt,mlVar);

close(rpt);
rptview(rpt);
```

### **NumericFormat — Format or precision used to display noninteger numeric values**

`[]` (default) | string scalar | character vector | positive integer

Format or precision used to display noninteger numeric values.

Specify a format as a string scalar or a character vector. See the `formatSpec` argument on the `sprintf` reference page.

Specify precision as a positive integer. See the `precision` argument on the `num2str` reference page.

Example: `"%.2f"` displays double values with two digits to the right of the decimal place.

Example: `2` displays a maximum number of two significant digits.

### **TemplateSrc — Source of template for this reporter**

`[]` (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

### **TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.



**LinkTarget — Hyperlink target for this reporter**

[] (default) | character vector | string scalar | mlreportgen.dom.LinkTarget object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an mlreportgen.dom.LinkTarget object. A character vector or string scalar value is converted to a LinkTarget object. The link target immediately precedes the content of this reporter in the output report.

**Methods**

createTemplate	Create MATLAB variable template
customizeReporter	Create custom MATLAB variable reporter class
getClassFolder	MATLAB variable class definition file location
getVariableValue	Get MATLAB variable value
setVariableValue	Set the value to report for a variable

**Inherited Methods**

copy	Create copy of reporter object and make deep copies of certain property values
getImpl	Get implementation of reporter

**Examples****Report Variables Using Direct Names or Strings**

This example shows how to report on MATLAB variables. The local variable uses only its name as input to the MATLABVariable class and the workspace variable uses a string. The first part of the example uses default property settings and the second part changes the display to a table.

---

**Note** Before you run this example, create this variable in the base MATLAB workspace:

```
workspace_var = ['Workspace variable input ',...
    'specified as a string'];

rpt = mlreportgen.report.Report("MyReport","pdf");

local_var = ['Local variable input specified ',...
    'using its variable name'];

chapter = mlreportgen.report.Chapter();
chapter.Title = "MATLAB Variable Reporter Example";

% Format using default paragraphs
rptr_local1 = mlreportgen.report.MATLABVariable...
    (local_var);
rptr_workspace1 = mlreportgen.report.MATLABVariable...
    ("workspace_var");

add(chapter,rptr_local1)
```

```

add(chapter, rptr_workspace1)

% Format as a table
rptr_local2 = mlreportgen.report.MATLABVariable...
    (local_var);
rptr_workspace2 = mlreportgen.report.MATLABVariable...
    ("workspace_var");
rptr_local2.FormatPolicy = 'Table';
rptr_workspace2.FormatPolicy = 'Table';

add(chapter, rptr_local2)
add(chapter, rptr_workspace2)
add(rpt, chapter)

close(rpt)
rptview(rpt)

```

### Chapter 1. MATLAB Variable Reporter Example

**local\_var.** Local variable input specified using its variable name

**workspace\_var.** Workspace variable input specified as a string

**Table 1.1. local\_var**

<b>Value</b>	Local variable input specified using its variable name
<b>Data Type</b>	char

**Table 1.2. workspace\_var**

<b>Value</b>	Workspace variable input specified as a string
<b>Data Type</b>	char

### See Also

`mlreportgen.dom.Text` | `mlreportgen.dom.Paragraph` | `mlreportgen.report.BaseTable`

**Introduced in R2018b**

# mlreportgen.report.Report class

**Package:** mlreportgen.report

Report container

## Description

An object of the `mlreportgen.report.Report` class is a container for a report based on reporters and MATLAB and DOM objects. Use an `mlreportgen.report.Report` object to generate an HTML, PDF, or Word report based on templates in a template library.

The `mlreportgen.report.Report` class is a `handle` class.

## Class Attributes

`HandleCompatible` `true`

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`report = mlreportgen.report.Report()` returns a report container object with default property values.

`report = mlreportgen.report.Report(path)` sets the `OutputPath` property to `path`.

`report = mlreportgen.report.Report(path,type)` also sets the “Type” on page 12-0 property to `type`.

`report = mlreportgen.report.Report(path,type,template)` also sets the `TemplatePath` property to `template`.

`report = mlreportgen.report.Report(Name=Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order.

## Properties

### OutputPath — Path of the generated report file

character vector | string scalar

Path of generated report file, specified as a character vector or string scalar. The path is the location in the file system where the report output document is stored. The path can be a full path, for example, `'C:/myreports/reportA.docx'`. The path can also be relative to the current MATLAB folder, for example, `'reportA'`. If the file name does not have a file extension that corresponds to the `Type` property, the appropriate file extension is added.

**Note** Generating a PDF report on a cloud drive, such as MATLAB Drive, can result in an error that is caused by file contention between the report generation software and the cloud drive synchronization

software. To avoid this error, generate reports on a local drive that does not synchronize with the cloud. Consider writing a script that generates a report on a local drive and then copies the report to the cloud drive.

### PackageType — Packaging for generated files

'zipped' | 'unzipped' | 'both' | 'single-file'

Packaging used for the generated files, specified as one of the values in the table.

Value	Supported Report Types	Description
'zipped'	'docx' or 'html'	Generates the report as a zip file at the location specified by the <code>OutputPath</code> property. The zip file has an extension that matches the document type ( <code>docx</code> for Word output or <code>html</code> for HTML output.) For example, if the document type is <code>docx</code> and <code>OutputPath</code> is <code>s:\docs\MyDoc</code> , the output is packaged in a zip file named <code>s:\docs\MyDoc.docx</code> .
'unzipped'	'docx' or 'html'	Generates the report as separate files in a folder that has the file name of the <code>OutputPath</code> property. For example, if the <code>OutputPath</code> is <code>s:\docs\MyDoc</code> , the output folder is <code>s:\docs\MyDoc</code> .
'both'	'docx' or 'html'	Generates zipped and unzipped outputs.
'single-file'	'pdf' or 'html-file'	Generates the report as a single file.

When the `Type` property is `'html'`, to generate an HTML report that you can open without unzipping, set `PackageType` to `'unzipped'` or `'both'`. In the folder that contains the generated files, open the `root.html` file.

### Type — Output type

'pdf' (default) | 'html' | 'html-file' | 'docx'

Output type, specified as one of these values:

- `'pdf'` - PDF file.
- `'html'` - HTML report, packaged as a zipped file that contains the HTML file, images, style sheet, and JavaScript files of the report. To generate an HTML report as a folder that contains unzipped files, set the `PackageType` property to `'unzipped'` or `'both'`.
- `'html-file'` - HTML report, consisting of a single HTML file that contains the text, style sheets, JavaScript, and base64-encoded images of the report.

- 'docx' - Microsoft Word document.

If you specify a template using the `TemplatePath` property, the value for `Type` must match the template type.

### Layout — Page layout options

`mlreportgen.report.ReportLayout` object

Page layout options for this report, specified as an `mlreportgen.report.ReportLayout` object. The initial value of the `Layout` property is an `mlreportgen.report.ReportLayout` object with default values. Customize the page layout by modifying the property values. For an example, see “Create a Landscape Report” on page 12-855.

The layout options specified by the `Layout` property of objects of the `mlreportgen.report.TitlePage`, `mlreportgen.report.TableOfContents`, and `mlreportgen.report.Chapter` classes can override the page layout properties specified by the `Layout` property of an `mlreportgen.report.Report` object.

---

**Note** The `Layout` property applies only to PDF and Word reports.

---

### Locale — Locale or language

[ ] (default) | character vector | string scalar

Locale or language, specified as a character vector or string scalar that consists of the ISO\_639-1 two-letter language code of the locale for which this report is to be generated. The default value, [ ], specifies the language of the system locale, for example, English on an English system. The Report API uses the language code to translate chapter title prefixes to the language of the specified locale. Translations are provided for the following locales: 'af', 'ca', 'cs', 'da', 'de', 'el', 'en', 'es', 'et', 'eu', 'fi', 'fr', 'hu', 'id', 'it', 'ja', 'ko', 'nl', 'nn', 'no', 'pl', 'pt', 'ro', 'ru', 'sk', 'sl', 'sr', 'sv', 'tr', 'uk', 'xh', and 'zh'. If you specify an unsupported locale, the English version is used. See [https://en.wikipedia.org/wiki/List\\_of\\_ISO\\_639-1\\_codes](https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes).

For an example, see “Translate Chapter Title Prefixes” on page 12-855.

### TemplatePath — Location of template

character vector | string scalar

Location of the template used to format this report, specified as a character vector or string scalar. You can use this property to specify a custom template for the report.

### Document — Underlying DOM document object

`mlreportgen.dom.Document`

Underlying DOM document object used to generate the content of the report, specified as an `mlreportgen.dom.Document` object. This property is read-only.

### Context — Report context

`containers.Map` object

Report context, specified as a `containers.Map` object that contains information to generate the report, such as the hierarchical level of the current report section. This property is read-only.

### Debug — Debug mode

false or 0 (default) | true or 1

Debug mode, specified as a numeric or logical 1 (`true`) or 0 (`false`). If you set `Debug` to `true` or 1, the temporary files for the report are stored in a subfolder of the report folder and are not deleted when the report is closed.

## Methods

### Public Methods

<code>open</code>	Opens the report
<code>append</code>	Add content to report
<code>add</code>	(Not recommended) Add content to report
<code>close</code>	Close and generate report
<code>rptview</code>	Open generated report file in viewer
<code>mlreportgen.report.Report.createTemplate</code>	Create report template
<code>mlreportgen.report.Report.customizeReport</code>	Create class derived from Report class
<code>mlreportgen.report.Report.getClassFolder</code>	Report class definition file location
<code>getTempPath</code>	Path of report temporary directory
<code>generateFileName</code>	Generate temporary report file name
<code>getReportLayout</code>	Current page layout of report
<code>fill</code>	Fill report template holes
<code>getContext</code>	Get report context value
<code>setContext</code>	Set report context value
<code>ispdf</code>	Check if PDF report
<code>isdocx</code>	Check if Word report
<code>ishtml</code>	Check if multifile HTML report
<code>ishtmlfile</code>	Check if single-file HTML report

## Examples

### Create a Report

Create a report using the Report API.

Import the DOM and Report API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
import mlreportgen.dom.*
```

Create the report container.

```
rpt = Report('My Report', 'pdf');
```

Add a title page, table of contents, and chapter to the report. The chapter contains two sections, each of which contains an image.

```
append(rpt, TitlePage(Title='My Report'));
append(rpt, TableOfContents);
ch = Chapter('Images');
append(ch, Section(Title='Boeing 747', ...
    Content=Image(which('b747.jpg'))));
append(ch, Section(Title='Peppers', ...
    Content=Image(which('peppers.png'))));
append(rpt, ch);
```

```
close(rpt);
rptview(rpt);
```

## Create a Landscape Report

Create a report that has landscape orientation by using the Report API.

Import the Report API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
```

Create a report container. In the `mlreportgen.report.ReportLayout` object that is assigned to the `Layout` property, set the `Landscape` property to `true`.

```
rpt = Report('myreport', 'pdf');
rpt.Layout.Landscape = true;
```

Add content to the report. Generate and view the report.

```
append(rpt, TitlePage(Title='My Landscape Report'));
append(rpt, TableOfContents);
append(rpt, Chapter(Title='Tests'));
append(rpt, Chapter(Title='Unit Tests'));
close(rpt);
rptview(rpt);
```

## Translate Chapter Title Prefixes

Translate chapter title prefixes to Japanese on an English system by setting the `Locale` property.

```
import mlreportgen.report.*
rpt = Report('Japanese Report');
rpt.Locale = 'ja';
house = char(23478); % Kanji character for house
append(rpt, Chapter(house));
close(rpt);
rptview(rpt);
```

## Compatibility Considerations

### add method is not recommended

*Not recommended starting in R2020b*

Starting in R2020b, use the `append` method instead of the `add` method to add content to objects of these Report API classes:

- `mlreportgen.report.Report`
- `mlreportgen.report.Chapter`
- `mlreportgen.report.Section`

To add content to a DOM API object, such as an `mlreportgen.dom.Paragraph` object, continue to use the `append` method of the DOM object. The advantage of using `append` to add content to Report API objects is that you use the same method name as you use to add content to DOM API objects.

There are no plans to remove the add methods of the Report, Chapter, or Section classes. Report API programs that use the add methods will continue to run.

To update existing code, replace the method name add with append as shown by the examples in the table.

Not Recommended	Recommended
<pre>import mlreportgen.report.* import mlreportgen.dom.*  rpt = Report("My Report","pdf"); ch = Chapter("My Chapter"); sect = Section("My Section"); para = Paragraph("My Content "); append(para,"more Content"); add(sect,para); add(ch,sect); add(rpt,ch);  close(rpt); rptview(rpt);</pre>	<pre>import mlreportgen.report.* import mlreportgen.dom.*  rpt = Report("My Report","pdf"); ch = Chapter("My Chapter"); sect = Section("My Section"); para = Paragraph("My Content "); append(para,"more Content"); append(sect,para); append(ch,sect); append(rpt,ch);  close(rpt); rptview(rpt);</pre>

## See Also

`mlreportgen.report.Reporter`

### Topics

“Create a Report Program” on page 13-3

“Create Report Container” on page 13-8

“Create a Report Generator” on page 2-2

“What Is a Reporter?” on page 1-3

### Introduced in R2017b



# mlreportgen.report.Reporter class

**Package:** mlreportgen.report

Superclass for MATLAB reporters

## Description

mlreportgen.report.Reporter is a superclass for reporter objects that generate content to add to a report. The format of a reporter is based on a template.

The mlreportgen.report.Reporter class is a handle class.

## Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

reporter = mlreportgen.report.Reporter() creates an empty reporter.

report = mlreportgen.report.Reporter(Name=Value) sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order.

## Properties

### TemplateSrc — Source of template for this reporter

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, TemplateSrc must be a Word reporter template. If the TemplateSrc property is empty, this reporter uses the default reporter template for the output type of the report.

**Attributes:**

GetAccess	public
SetAccess	public

**TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template specified by the `TemplateSrc` property of this reporter.

**Attributes:**

GetAccess	public
SetAccess	public

**LinkTarget — Hyperlink target for this reporter**

[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

**Attributes:**

GetAccess	public
SetAccess	public

**Methods****Public Methods**

<code>copy</code>	Create copy of reporter object and make deep copies of certain property values
<code>getImpl</code>	Get implementation of reporter
<code>mlreportgen.report.Reporter.createTemplate</code>	Create reporter template
<code>mlreportgen.report.Reporter.customizeReporter</code>	Create class derived from Reporter class
<code>mlreportgen.report.Reporter.getClassFolder</code>	Get location of folder containing <code>mlreportgen.report.Reporter</code> class definition file

**See Also**

`mlreportgen.report.Report` | `mlreportgen.report.TitlePage` |  
`mlreportgen.report.TableOfContents` | `mlreportgen.report.Section` |  
`mlreportgen.report.Chapter` | `mlreportgen.report.Equation` |  
`mlreportgen.report.Figure` | `mlreportgen.report.FormalImage` |  
`mlreportgen.report.ReporterLayout` | `mlreportgen.report.ReportLayout` |  
`mlreportgen.report.InlineContent` | `mlreportgen.report.BaseTable`

**Topics**

“What Is a Reporter?” on page 1-3  
 “Define a New Reporter” on page 1-8

**Introduced in R2017b**

## mlreportgen.report.ReporterLayout class

**Package:** mlreportgen.report

Layout for reporter

### Description

Use an object of the `mlreportgen.report.ReporterLayout` class to specify page layout options for a report section, such as a title page, table of contents, list of figures, list of tables, list of captions, or chapter. Page layout options include watermarks, first page numbering, page number formatting, page size and orientation, and margins.

---

**Note** When you create the reporter for a report section, an instance of this class is created and assigned to the reporter `Layout` property. You do not create this object yourself.

---

The `mlreportgen.report.ReporterLayout` class is a handle class.

### Class Attributes

`HandleCompatible` true

For information on class attributes, see “Class Attributes”.

### Properties

#### Watermark — Watermark image

`[]` (default) | string scalar | character vector

Watermark image for the report section pages, specified as `[]` or a character vector or string scalar that contains the image path name. The watermark appears on all pages of the report section. If the `Watermark` property is `[]`, the watermark for the section is the same as the watermark for the report. If the `Watermark` property of the layout objects for the report section and the report are `[]`, no watermark appears.

Valid image types are:

- `.bmp`
- `.jpg`
- `.pdf` (for PDF output types only)
- `.png`
- `.svg`
- `.tiff`

#### FirstPageNumber — Number to use on first page of section

`[]` (default) | integer

Number to use on the first page of the report section, specified as an integer or []. If the value of this property is -1 or [], the numbering is continued from the previous section. To specify the first page number, enter a positive integer.

---

**Note** By default, the number of the first page of the first chapter is 1.

---

### PageNumberFormat — Type of page numbering

[] (default) | string scalar | character vector

Type of page numbering to use for the report section in a Word or PDF report, specified as a string scalar or character vector. See the `Format` property in `mlreportgen.dom.PageNumber` for a list of valid page number formats.

### Landscape — Page orientation for section

[] (default) | true | false

Page orientation for the report section, specified as `true` or `false`. Set this property to `true` to change portrait orientation to landscape orientation and `false` to change landscape orientation to portrait orientation. If the value is [], the orientation is determined by the page size specified by the section layout.

### PageSize — Size of pages

[] (default) | `mlreportgen.dom.PageSize` object

Size of pages in the report section, specified as an `mlreportgen.dom.PageSize` object.

### PageMargins — Sizes of page margins, header, footer, and gutter

[] (default) | `mlreportgen.dom.PageMargins` object

Sizes of page margins, header, footer, and gutter in the report section, specified as an `mlreportgen.dom.PageMargins` object.

### PageBorder — Page borders

[] (default) | `mlreportgen.dom.PageBorder` object

Page borders, specified as an `mlreportgen.dom.PageBorder` object.

## Examples

### Set First Page Number and Page Orientation for a Chapter

Add three chapters to a report. The first chapter uses default values for the first page number, which is 1 and the default page orientation which is portrait. The second chapter resets the first page number to 1 and uses landscape page orientation. The page number and orientation are not set in the third chapter so it uses the default first page number, which continues from the previous chapter, and the default page orientation, which is portrait.

```
import mlreportgen.report.*
rpt = Report('newreport');

tp = TitlePage();
tp.Title = 'New Report';
```

```
tp.Author = 'MathWorks';
add(rpt,tp)

ch1 = Chapter();
ch1.Title = 'First Chapter';
sec = Section('First Section of Chapter 1');
txt = ['This is the first section of chapter 1. ',...
      'The first page number for this ',...
      'chapter is 1, which is the default. ',...
      'The page orientation is also the default.'];
append(sec,txt);
append(ch1,sec);
append(rpt,ch1);

ch2 = Chapter();
ch2.Title = '2nd chapter';
ch2.Layout.FirstPageNumber = 1;
ch2.Layout.Landscape = true;
sec = Section('First Section of Chapter 2');
txt = ['This is the first section of chapter 2. ',...
      'The first page number is set to 1 and the ',...
      'page orientation is set to landscape.'];
append(sec,txt);
append(ch2,sec);
append(rpt,ch2);

ch3 = Chapter();
ch3.Title = '3rd chapter';
sec = Section('First Section of Chapter 3');
txt = ['This is the first section of chapter 3. ',...
      'Neither first page number nor page ',...
      'orientation is set for this chapter. ',...
      'The first page number uses the default, ',...
      'which continues from the previous page. ',...
      'The page orientation also uses the default, ',...
      'which is portrait.'];
append(sec,txt);
append(ch3,sec);
append(rpt,ch3);

close(rpt);
rptview(rpt)
```

## See Also

[mlreportgen.dom.PageNumber](#) | [mlreportgen.report.ReportLayout](#) |  
[mlreportgen.report.Report](#) | [mlreportgen.report.Reporter](#) |  
[mlreportgen.dom.PageSize](#) | [mlreportgen.dom.PageMargins](#) |  
[mlreportgen.dom.PageBorder](#)

## Topics

“What Is a Reporter?” on page 1-3

“Customize the Page Size and Margins of a Report Programmatically” on page 17-154

## Introduced in R2017b

# mlreportgen.report.ReportLayout class

**Package:** mlreportgen.report

Page layout of report

## Description

Use an object of the mlreportgen.report.ReportLayout class to specify report page layout options, such as watermarks, first page numbering, page number formatting, page size and orientation, and margins.

---

**Note** A report creates an instance of this class and assigns it to the mlreportgen.report.ReportLayout property. You do not create the object yourself.

---

The mlreportgen.report.ReportLayout class is a handle class.

## Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Properties

### Watermark — Watermark image

[ ] (default) | string scalar | character vector

Watermark image for the report pages, specified as [ ] or a character vector or string scalar that contains the image path name. The specified watermark appears on all pages of the report. An empty value, [ ], indicates that no watermark is used. To use a different watermark for an individual report section, such as a chapter, set the Watermark property of the mlreportgen.report.ReporterLayout object used by the section reporter. Valid image types are:

- .bmp
- .jpg
- .pdf (for PDF output types only)
- .png
- .svg
- .tiff

### FirstPageNumber — Number to use on first page

[ ] (default) | positive integer

Number to use on the first page of each section in the report, specified as [ ] or a positive integer. For example, if you set the first page number for the report to 4, the first page number for every report chapter is 4. To use a different first page number for an individual section, set the FirstPageNumber property of the mlreportgen.report.ReporterLayout object used by the section reporter. The

default numbering for the report is `[]`, which indicates that the first page of chapter 1 is page 1. All subsequent pages in the report are numbered sequentially.

### **PageNumberFormat — Type of page numbering**

`[]` (default) | string scalar | character vector

Type of page numbering to use for the report, specified as a string scalar or character vector. The specified page number format appears on all pages of the report. To use a different page number format for an individual section, such as a chapter, set the `PageNumberFormat` property of the `mlreportgen.report.ReporterLayout` object used by the section reporter. See the `Format` property in `mlreportgen.dom.PageNumber` for a list of valid page number formats.

### **Landscape — Page orientation**

`[]` (default) | `false` | `true`

Page orientation for the report, specified as `true` or `false`. Set this property to `true` to change portrait orientation to landscape orientation and `false` to change landscape orientation to portrait orientation. If the value is `[]`, the orientation is determined by the page size specified by the report layout.

### **PageSize — Size of Pages**

`[]` (default) | `mlreportgen.dom.PageSize` object

Size of pages in the report, specified as an `mlreportgen.dom.PageSize` object.

### **PageMargins — Sizes of page margins, header, footer, and gutter**

`[]` (default) | `mlreportgen.dom.PageMargins` object

Sizes of page margins, header, footer, and gutter in the report, specified as an `mlreportgen.dom.PageMargins` object.

### **PageBorder — Page borders**

`[]` (default) | `mlreportgen.dom.PageBorder` object

Page borders, specified as an `mlreportgen.dom.PageBorder` object.

## **Examples**

### **Set First Page Number for Entire Report**

Set the page number format for the whole report to Arabic numbers and the page number for the table of contents to Roman numerals. The chapters use the Arabic number format, which is the default format for the whole report. The first page of the first chapter defaults to 1.

```
import mlreportgen.report.*
rpt = Report('newreport');
rpt.Layout.PageNumberFormat = 'n';

tp = TitlePage();
tp.Title = 'New Report';
tp.Author = 'MathWorks';
append(rpt,tp);

toc = TableOfContents();
```



```
toc.Layout.PageNumberFormat = 'i';
append(rpt,toc);

ch = Chapter();
ch.Title = 'Introduction';
sec = Section('First Section of Chapter 1');
txt = ['This is the first section of chapter 1. ',...
       'The page number format is Arabic numbers, ',...
       'which is the default for the report.'];
append(sec,txt);
append(ch,sec);
append(rpt,ch);

ch = Chapter();
ch.Title = '2nd chapter';
sec = Section('First Section of Chapter 2');
txt = ['This is the first section of chapter 2. ',...
       'The page number format is Arabic numbers, ',...
       'which is the format defined for the report.'];
append(sec,txt);
append(ch,sec);
append(rpt,ch);

close(rpt);
rptview(rpt);
```

## See Also

[mlreportgen.dom.PageNumber](#) | [mlreportgen.report.ReporterLayout](#) |  
[mlreportgen.report.Report](#) | [mlreportgen.report.Reporter](#) |  
[mlreportgen.dom.PageSize](#) | [mlreportgen.dom.PageMargins](#) |  
[mlreportgen.dom.PageBorder](#)

## Topics

“What Is a Reporter?” on page 1-3

“Customize the Page Size and Margins of a Report Programmatically” on page 17-154

## Introduced in R2017b

## mlreportgen.report.RptFile class

**Package:** mlreportgen.report

**Superclasses:** mlreportgen.report.Reporter

Create Report Explorer-based reporter

### Description

Use the `RptFile` reporter to include the content generated by a Report Explorer setup (`.rpt`) file in a Report API report. When added to a report, the `RptFile` reporter:

- 1 Executes the specified Report Explorer setup file to generate a DocBook XML rendition of the Report Explorer report
- 2 Uses a modified version of the Report Explorer Docbook-to-DOM conversion template to convert the XML to a set of DOM objects (see “Manage Report Conversion Templates”)
- 3 Adds the DOM content to the Report API report.

The `mlreportgen.report.RptFile` class is a `handle` class.

### Creation

#### Description

`reporter = RptFile()` creates an empty Report Explorer-based `RptFile` reporter. Before adding the reporter to a report, your report program must set the reporter's `SetupFile` property to the path of a Report Explorer setup (`.rpt`) file. Otherwise, an error occurs.

By default the `RptFile` reporter uses a conversion template that is a slightly modified version of the Report Explorer's default conversion template for the report output type. For example, if the report output type is PDF, the reporter uses a slightly modified version of the default template for the Report Explorer's PDF (`from template`) output type.

You can use a custom conversion template to customize the reporter output. Use the reporter's `createTemplate` method to create a copy of one of the reporter's default output-type-specific conversion templates for customization. To use the customized template, set the `RptFile` reporter's `TemplateSrc` property to the path of the customized template.

`reporter = RptFile(SetupFile)` creates a `RptFile` reporter based on the specified Report Explorer setup file (`.rpt` file). See the `SetupFile` property.

`reporter = RptFile(Name=Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order.

### Properties

#### **SetupFile — Report Explorer setup file path**

character array | string

Report Explorer setup file path, specified as a character array or string. Do not use form-based reports for setup files that you use with the `RptFile` reporter.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

Data Types: `character array | string`

**TemplateSrc — Source of conversion template**

`[] | string | character array`

Source of conversion template to be used by this reporter to convert the setup file's XML output to DOM objects. An empty value specifies use of the default template for the output type of the report to be generated. A string or character array value specifies the path of a customized version of the default template for the output type to be generated.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

Data Types: `character array | string`

**TemplateName — Name of template for this reporter**

`character array | string`

Name of template for this reporter, specified as a character array or string. By default this property specifies `RptFile`, the name of the reporter's default template. This default template resides in the template library of its default conversion template along with other templates used to convert Report Explorer XML components to DOM objects. The default reporter template contains a single hole named `Content` to be filled with the DOM content converted from the XML content generated by the setup. If you change the name of this template, you must set this property to the new name. You can modify the template itself, but the modified template must contain a hole named `Content`.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

Data Types: `character array | string`

**LinkTarget — Hyperlink target for content created by this reporter**

`character array | string | mlreportgen.dom.LinkTarget object`

Hyperlink target for this reporter, specified as a character array or string that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A string or character array value is converted to a `LinkTarget` object. The link target object immediately precedes the content of this reporter in the output report.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

Data Types: `character array | string | object`

## Methods

### Public Methods

<code>createTemplate</code>	Create Report Explorer-based (RptFile) reporter template
<code>customizeReporter</code>	Create custom Report Explorer-based reporter class
<code>getClassFolder</code>	Report Explorer-based reporter class definition file location
<code>copy</code>	Create copy of reporter object and make deep copies of certain property values
<code>getImpl</code>	Get implementation of reporter

## Examples

### Create a RptFile Reporter

Create an `RptFile` reporter without specifying a setup file. Then, use the `SetupFile` property to specify the Report Explorer setup file.

```
reporter = mlreportgen.report.RptFile();  
reporter.SetupFile = "my_setup_file.rpt"
```

### Add Syntax-Highlighted Code to a Report

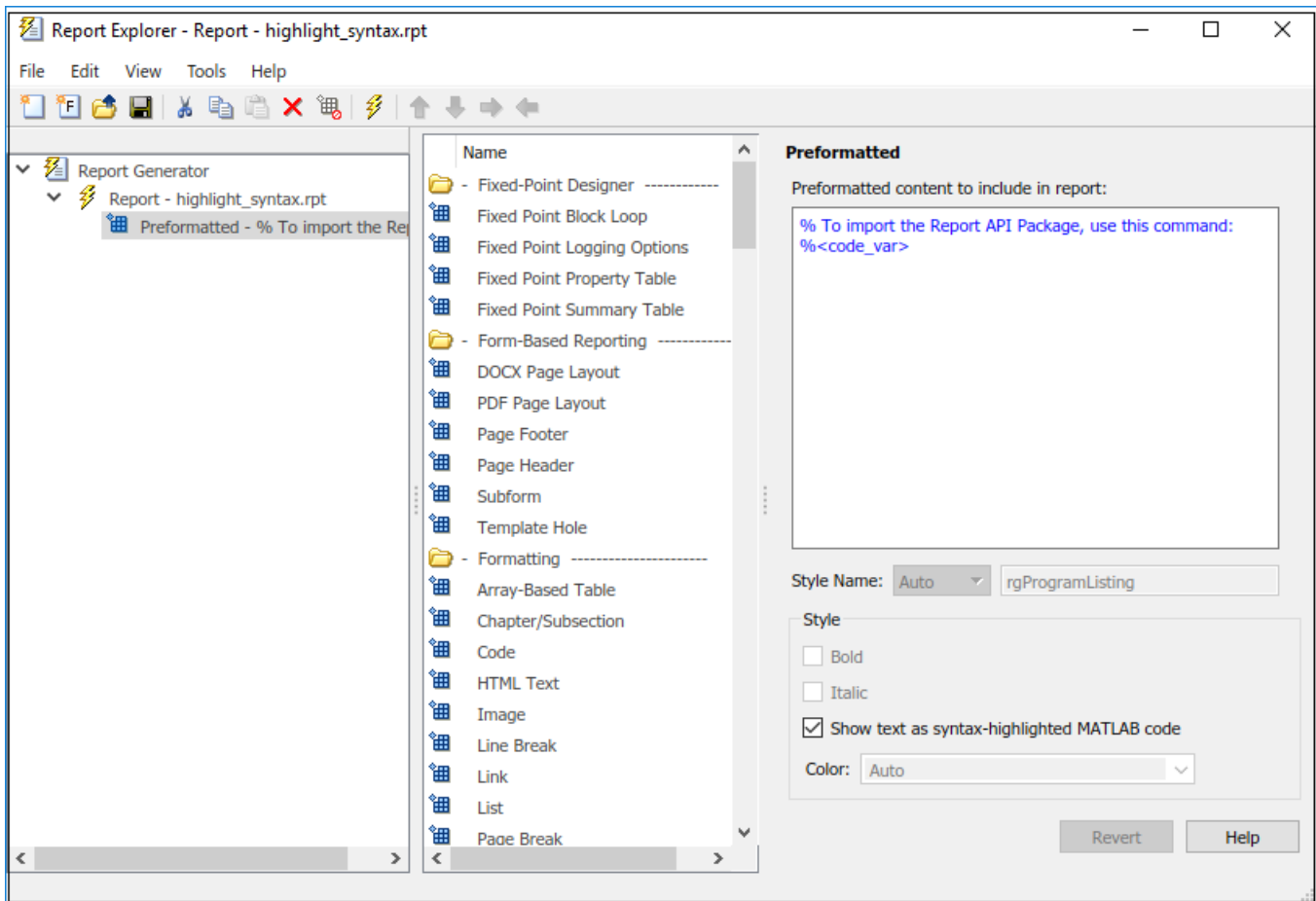
Use the `RptFile` reporter to add syntax-highlighting to code in a Report API Report.

---

**Note** Before you run this example, use the Report Explorer to create a setup file named `highlight_syntax.rpt`. The setup file for this example contains a Preformatted component with this text in its edit box:

```
% To import the Report API Package, use this command:  
%<code_var>
```

Enable **Show text as syntax-highlighted MATLAB code**, which is below the Preformatted text box. See "Create a Report Setup File" on page 2-10.



This code creates a Report API report that includes the `highlight_syntax.rpt` setup file.

```
rpt = mlreportgen.report.Report("My Report", "pdf");

chap = mlreportgen.report.Chapter...
    ("Include Report Explorer Report Using the RptFile Reporter");
sect1 = mlreportgen.report.Section...
    ("Highlighted Syntax Example");

% Evaluate the expression and assign it to the code variable
evalin('base', 'code_var = "import mlreportgen.report.*"');

rptfile = mlreportgen.report.RptFile("highlight_syntax.rpt");

add(sect1, rptfile)
add(chap, sect1)
add(rpt, chap)

close(rpt)
rptview(rpt)
```

---

## Chapter 1. Include Report Explorer Report Using the RptFile Reporter

### 1.1. Highlighted Syntax Example

```
% To import the Report API Package, use this command:  
import mlreportgen.report.*
```

### See Also

`mlreportgen.report.Reporter` | `mlreportgen.report.Report`

### Topics

“Add Report Explorer Content to a Report” on page 3-16

“Working with the Report Explorer” on page 1-16

“Add Report Explorer Content to a Report” on page 3-16

### Introduced in R2019a

# mlreportgen.report.Section class

**Package:** mlreportgen.report

**Superclasses:** mlreportgen.report.Reporter

Section reporter

## Description

Create a section reporter that adds a section to the report. This class inherits from mlreportgen.report.Reporter

The mlreportgen.report.Section class is a handle class.

## Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

`section = Section()` creates a reporter that generates a report section. You can add the section reporter to a report, chapter, or another section. If you add a section to a report, the section starts on a new, portrait page with default margins and a page number in the footer. The page number equals the previous page number plus one. If you add the section to a chapter or another section, the reporter creates a subsection that continues on the current page. The size of the title diminishes by default with the depth of the section in the report hierarchy up to five levels deep. Titles of sections lower than 5 are not numbered and have the same font size as level 5.

`section = Section(title)` creates a report section containing a section title with the specified title text. A hierarchical section number prefixes the title text by default. For example, the default number of the first subsection in the second chapter is 2.1. The font size of the title diminishes by default with the depth of the section in the report hierarchy up to five levels deep.

`section = Section(Name=Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order.

## Properties

### Title — Section title

string scalar | string array | character array | DOM object array | cell array

Section title, specified as one of these values:

- string scalar or character array
- DOM object

- 1-by-*N* or *N*-by-1 array of strings or DOM objects
- 1-by-*N* or *N*-by-1 cell array of strings, character arrays, and/or DOM objects
- `SectionTitle` reporter

Inline objects are objects that a paragraph can contain. If the title value is an inline object, the section object uses one template from a set of templates. Templates are stored in the template library for the section. The template used to create the title depends on whether the title is numbered and the section level in the section hierarchy. Use the `Numbered` property to specify whether the section title is numbered.

If the title value is a DOM paragraph or other DOM block object, the section inserts the object at the beginning of the section. If you use a DOM block object, you can use block elements to customize the spacing, alignment, and other properties of the section title. In this case, you must fully specify the title format and provide title numbering yourself.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

Data Types: `string` | `char` | `DOM object` | `cell`

**Numbered — Option to number this section**

`[]` (default) | `logical true` or `1` | `false` or `0`

Choice to number this section, specified as a `logical`. If the value of this property is `[]` or `true`, the section is numbered relative to other sections in the report. The section number appears in the section title. If the value is `false`, this section is not numbered. The value of this `Numbered` property overrides the numbering specified for all report sections by the `mlreportgen.report.Section.number` method.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

Data Types: `logical`

**Content — Content of this section**

`string scalar` | `character array` | `inline object`

Content of the section, specified as one of these values:

- String or character array
- DOM objects that can be added to a DOM document part
- Reporters, including `Section` reporters
- 1x*N* or *N*x1 array of strings or character arrays
- 1x*N* or *N*x1 cell array of strings, character arrays, and/or DOM objects

Use the `Section` constructor or `add` method to set this property. You cannot set it directly.



**Attributes:**

GetAccess	public
SetAccess	public

**TemplateSrc — Source of template for this reporter**

[ ] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

**TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

**LinkTarget — Hyperlink target for this reporter**

[ ] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

**Methods****Public Methods**

add	(Not recommended) Add content to section
append	Add content to section
createTemplate	Create section template
customizeReporter	Create custom section reporter class
getClassFolder	Section class definition file location
getTitleReporter	Create a section title reporter
number	Set section numbering

## Examples

### Add Content to a Report Section

Add a section to a chapter and the chapter to a report. Set the layout orientation of the chapter to landscape.

```
import mlreportgen.report.*
import mlreportgen.dom.*

theReport = Report("SectionExampleReport","pdf");

append(theReport,TitlePage(Title="Report with Sections"));

append(theReport,TableOfContents);

theChapter = Chapter("Images");
append(theChapter,Section(Title="Boeing 747",Content=Image("BoeingSectionExample.jpg")));
append(theChapter,Section(Title="Peppers",Content=Image("PeppersSectionExample.png")));
append(theReport,theChapter);

close(theReport);
rptview(theReport);
```

Here are the sections with the images in the generated report.

## Chapter 1. Images

### 1.1. Boeing 747



Chapter 1. Images

---

1.2. Peppers



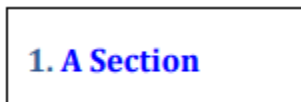
## Use DOM Text Object as a Section Title

This example uses a DOM Text object to define the title. By using the DOM object, you can set its properties and override the default black color of the section title.

```
import mlreportgen.report.*
import mlreportgen.dom.*

rpt = Report('New Report', 'pdf');
open(rpt)
sect = Section;
sect.Title = Text('A Section');
sect.Title.Color = 'blue';
append(rpt, sect);

close(rpt)
rptview(rpt)
```



## Change Alignment of a Section

This example generates a report that sets the subsection titles to center alignment.

```
import mlreportgen.report.*
import mlreportgen.dom.*

rpt = Report('My Report', 'html');
append(rpt, TitlePage(Title='My Report'));
append(rpt, TableOfContents);
chTitle = Heading1('Chapter ');
chTitle.Style = {CounterInc('sect1'), ...
    WhiteSpace('preserve') ...
    Color('black'), ...
    Bold, FontSize('24pt')};
append(chTitle, AutoNumber('sect1'));
append(chTitle, '. ');

sectTitle = Heading2();
sectTitle.Style = {CounterInc('sect2'), ...
    WhiteSpace('preserve') ...
    HAlign('center'), PageBreakBefore};
append(sectTitle, AutoNumber('sect1'));
append(sectTitle, '. ');
append(sectTitle, AutoNumber('sect2'));
append(sectTitle, '. ');
title = clone(chTitle);
append(title, 'Images');
ch = Chapter(Title=title);
title = clone(sectTitle());
append(title, 'Boeing 747');
append(ch, Section(Title=title, Content=Image(which('b747.jpg'))));
title = clone(sectTitle());
append(title, 'Peppers');
```

```
append(ch,Section(Title=title,Content=Image(which('peppers.png'))));  
  
append(rpt,ch);  
close(rpt);  
rptview(rpt);
```

## Chapter 1. Images

### 1.1. Boeing 747

#### **See Also**

`mlreportgen.report.Reporter` | `mlreportgen.report.Report` | “What Is a Reporter?” on page 1-3 | “Templates for DOM API Report Programs” on page 13-22 | “Define a New Reporter” on page 1-8 | “Subclass a Reporter Definition” on page 1-13

**Introduced in R2017b**

# mlreportgen.report.SectionTitle class

**Package:** mlreportgen.report

Generate a section title

## Description

Only another report, called the owner, can create an instance of this class. The instance fills a hole in a template that the owner specifies with a title that the owner specifies via an instance template and the instance properties that it sets. For example, an `mlreportgen.report.Section` reporter (the owner) uses an instance of this class to fill the `Title` hole in its template with a hierarchically numbered title.

The title generated by an instance of this class consists of static text and dynamic content. The static text is specified by the instance template. The dynamic content is specified by the instance `Content`, `NumberPrefix`, and `NumberSuffix` properties. If the name of a template hole matches one of these property names, this reporter fills the hole with the value property. For example, if the title template contains a `Content` hole and a `NumberPrefix` hole, this reporter fills those holes with the values of the corresponding properties.

The owner specifies the location of the title template with the instance `TemplateSrc`, `TemplateName`, and `OutlineLevel` properties. The `TemplateSrc` property specifies the source of a template file whose template library contains the title template. The `TemplateName` property specifies the root name of the title template. The `OutlineLevel` property specifies the level in the report hierarchy of the content generated by the owner. The `SectionTitle` instance appends the value of the `OutlineLevel` property to the value of the `TemplateName` property to create the full name of the title template. For example, the `Section` reporter template library contains six numbered title templates, each corresponding to a section level. The names of the templates are `SectionNumberedTemplate1`, `SectionNumberedTemplate2`, and so on. To specify use of the `SectionNumberedTemplate1` template to generate a section title, the `Section` reporter sets the `SectionTitle` `TemplateName` property to `SectionNumberedTemplate` and the `OutlineLevel` property to 1.

If the `SectionTitle` `LinkTarget` property is set, the instance precedes the generated title with a link target having the specified target ID.

## Properties

### **HoleId** — Name of title hole in owner template

character vector | string scalar

Name of the title hole in the owner template, specified as a character vector or string scalar.

### **Content** — Title text

string scalar | character vector | DOM object | 1-by-*N* or *N*-by-1 array of strings or DOM objects | 1-by-*N* or *N*-by-1 cell array of strings, character arrays, and/or DOM objects | ...

Title text, specified as one of these values:

- String scalar or character vector

- DOM object
- 1-by-*N* or *N*-by-1 array of strings or DOM objects
- 1-by-*N* or *N*-by-1 cell array of strings, character arrays, and/or DOM objects

**NumberPrefix — Prefix for a section number in the title**

string scalar | character vector

Prefix for a section number in the title, specified as a string scalar or character vector. If no prefix is specified, the default title prefix, translated based on the report locale, is used.

**NumberSuffix — Suffix for a section number in the title**

string scalar | character vector

Suffix for a section number in the title, specified as a string or character array. If no suffix is specified, the default title suffix, translated based on the report locale, is used.

**Translations — Translation map for the section title prefix and suffix**

structure

Translation map for the section title prefix and suffix, specified as a MATLAB structure. If the specified translation map does not contain a translation for the report locale, the `Translations` property uses `en` as the backup locale. See the `Locale` property of `mlreportgen.report.Report` for information about valid locales.

**OutlineLevel — Outline level of owner reporter**

integer in the range [1,6]

Outline level of the owner reporter in the report section hierarchy, specified as a positive integer in the range [1,6]. The `OutlineLevel` property specifies the level of the section generated by the owner reporter in the section hierarchy of a report. The `SectionTitle` reporter appends the value of this property to the value of the `TemplateName` property to create the full name of the title template. For example, if the value of the `TemplateName` property is `SectionNumberedTitle` and the value of the `OutlineLevel` property is 2, this reporter generates `SectionNumberedTitle2` as the full name of the title template. The template library specified by the `TemplateSrc` property must contain a template with the generated name. Otherwise, an error occurs.

**TemplateSrc — Source of template for this reporter**

character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.



**TemplateName — Template root name**

character vector | string scalar

Template root name, specified as a character vector or string scalar. The template for this reporter is in the template library of the template source (`TemplateSrc`) for this reporter.

**LinkTarget — Hyperlink target for this reporter**[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

**See Also**

`mlreportgen.report.Report` | `mlreportgen.report.Reporter` |  
`mlreportgen.report.Chapter` | `mlreportgen.report.Section`

**Topics**

“What Is a Reporter?” on page 1-3

**Introduced in R2018b**

## mlreportgen.report.SummaryTable class

**Package:** mlreportgen.report

Summarize finder results

### Description

Use an instance of mlreportgen.report.SummaryTable class to generate a table that summarizes the results of a search by a MATLAB Report Generator finder object. The table lists the key properties of each finder result object. For more information, see “finder result class” on page 12-885.

The mlreportgen.report.SummaryTable class is a handle class.

### Class Attributes

HandleCompatible true

For information on class attributes, see “Class Attributes”.

## Creation

### Description

summaryTableRptr = mlreportgen.report.SummaryTable() creates an empty summary table reporter. Use the FinderResults property to specify the results to summarize.

---

**Note** Adding an empty MATLAB summary table reporter to a report causes an error.

---

summaryTableRptr = mlreportgen.report.SummaryTable(finderResults) creates a summary table reporter for the result objects in the array finderResults and sets the FinderResults property to finderResults.

summaryTableRptr = mlreportgen.report.SummaryTable(Name=Value) sets properties by using name-value arguments. Specify multiple name-value arguments in any order.

## Properties

### FinderResults — Finder result objects to report

finder result objects

Finder result objects to report, specified as an array of objects of a “finder result class” on page 12-885 such as mlreportgen.finder.MATLABVariableResult or mlreportgen.finder.AxesResult.

**Attributes:**

GetAccess	public
SetAccess	public

**Title — Title for summary table**

string scalar | character vector | DOM object

Title for the summary table, specified as a string, character vector, or a DOM object. By default, this property is empty and the reporter creates the table title by using the `getDefaultSummaryTableTitle` method for the finder result object specified by `finderResults`.

Specifying this property as a string scalar or character vector creates an `mlreportgen.dom.Paragraph` object with `Style` property set to 'SummaryTableTitleParagraph'.

**Attributes:**

GetAccess	public
SetAccess	public

**Properties — Properties of result class to report**

array | cell array

Properties of the result class to report on, specified as an array of string scalars or cell array of character vectors.

If `Properties` is empty, the reporter determines which finder result object properties to report by using the `getDefaultSummaryProperties` method of the finder result object specified by `finderResults`.

The summary table contains a row for each result whose entries contain properties of the found object. By default, the reported properties depend on the result object type. Use the summary table reporters `Properties` properties to customize the set of properties to report on.

Use the `getPropertyValues` method of the finder result object specified by `finderResults` to identify the properties you can add to the summary table.

**Attributes:**

GetAccess	public
SetAccess	public

**IncludeLinks — Whether to link to corresponding result object**

true (default) | false

Whether to link the `Title` property of each object in the summary table to link to the corresponding reporter object, specified as `true` or `false`.

**Attributes:**

GetAccess	public
SetAccess	public

**ShowEmptyColumns — Whether to include columns for empty result properties**

false (default) | true

Whether to include columns for empty finder result object properties in the summary tables, specified as `false` or `true`.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**TableReporter — Table reporter formatter**

`mlreportgen.report.BaseTable` object

Table reporter formatter type, specified as an `mlreportgen.report.BaseTable` object. Customize the appearance of the summary table by customizing the default reporter or by replacing it with a customized `BaseTable` reporter.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**Methods**

**Public Methods**

<code>mlreportgen.report.SummaryTable.createTemplate</code>	Create summary table reporter template
<code>mlreportgen.report.SummaryTable.customizeReporter</code>	Create class derived from summary table reporter class
<code>mlreportgen.report.SummaryTable.getClassFolder</code>	Get location of folder containing <code>mlreportgen.report.SummaryTable</code> class definition file
<code>getImpl</code>	Get implementation of reporter

**Examples**

**Add Summary Table Reporter to Report**

This example shows how to add a summary table for the results of an `mlreportgen.report.AxesFinder` object.

Create a PDF report.

```
import mlreportgen.report.*
import mlreportgen.finder.*

rpt = mlreportgen.report.Report("Summary Table Results","pdf");
add(rpt,TitlePage("Title",sprintf("Summary Table for Axes Finder Results"),"Author",""));
```

Create a figure handle and the axes to report.

```
figH = figure;
ax1 = subplot(3,2,2);
x = linspace(0,20);
y1 = cos(x);
plot(x,y1);
ax1.Title = title("Example Plot 1");

ax2 = subplot(3,2,4);
```

```
y2 = sin(tan(x).*x);  
plot(x,y2);  
ax2.Title = title("Example Plot 2");
```

Create an `AxesFinder` object to find the axes.

```
finderObj = mlreportgen.finder.AxesFinder(figH);  
results = find(finderObj);
```

Pass the results of the `find` method to a `SummaryTable` reporter.

```
rptr = mlreportgen.report.SummaryTable(results);  
rptr.ShowEmptyColumns = true;  
append(rpt,rptr);  
close(rpt);  
rptview(rpt);
```

## More About

### finder result class

A finder result class is a Report API class that is a subclass of the `mlreportgen.finder.Result` class, which is an undocumented, internal class.

## See Also

`mlreportgen.finder.AxesFinder` | `mlreportgen.finder.AxesResult`

### Introduced in R2022a

## **mlreportgen.report.TableOfContents class**

**Package:** mlreportgen.report

**Superclasses:** mlreportgen.report.Reporter

Table of contents reporter

### **Description**

Create a table of contents (TOC) reporter that adds a table of contents to the report. This class inherits from `mlreportgen.report.Reporter`.

### **Construction**

`toc = TableOfContents()` returns a reporter that generates a table of contents (TOC) section for the report. The default template for the TOC section defines the appearance and page layout of the TOC. The TOC section contains a default title and a TOC element that specifies the location of a TOC to be generated, depending on the report output type. The way in which the TOC is generated differs for each report type.

- **HTML** — JavaScript copied from the report template to the report generates the TOC when the report is opened in a browser. The script generates the TOC as a collapsible tree. The tree entries are the hyperlinked contents of the HTML heading elements (h1-h6) of the report. The level of an entry in the TOC tree corresponds to the level of the heading element. Chapter and section reporters generate chapter and section titles as heading elements of the appropriate level, so chapter and section titles appear automatically in the TOC. You can also use DOM Heading elements in a report to generate TOC entries.
- **DOCX** — The Report Generator `rptview` function instructs Word to generate the TOC after it opens the report in Word. If you open a report in Word directly, without using `rptview`, you must update the report document yourself to generate the TOC. See “Update a Table of Contents or Generated List in a Word Document” on page 3-37.

The TOC is a two-column table. The first column contains the hyperlinked contents of report paragraphs whose outline levels have been set. The outline level determines the formatting of a TOC entry. The second column contains the number of the page on which the corresponding paragraph occurs. Chapter and section reporters generate chapter and section titles as paragraphs with the appropriate level set, so chapter and section titles appear automatically in the TOC. You can also use DOM Heading elements in a report to generate TOC entries.

- **PDF** — The table of contents is generated during PDF document generation.

`toc = TableOfContents(title)` creates a TOC that uses the specified `title`.

`toc = TableOfContents(Name=Value)` sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order.

### **Properties**

#### **Title — Table of contents title**

Table of Contents (default) | string | character array | ...

Table of contents title, specified as one of these values:

- String or character array
- DOM object
- 1-by-*N* or *N*-by-1 array of strings or DOM objects
- 1-by-*N* or *N*-by-1 cell array of strings, character arrays, and/or DOM objects
- `TableOfContentsTitle` reporter

Example: `'TableOfContents', 'Report Contents'`

### **NumberOfLevels — Number of heading levels**

3 (default) | positive integer

Number of heading levels to use in the table of contents, specified as a positive integer in the range [1,9].

### **LeaderPattern — Type of leader**

'.' (default) | 'dots' | ' ' | 'space'

Type of leader to use between the title and the page number, specified as one of these character vectors or string scalars:

- '.' or 'dots'
- ' ' or 'space'

This property applies only to PDF reports. Word reports always have a dots leader. HTML reports do not have a leader.

### **Layout — Page layout for table of contents section**

`mlreportgen.report.ReporterLayoutobject`

Page layout for the table of contents section, specified as an `mlreportgen.report.ReporterLayout` object. Use the properties of the `ReporterLayout` object to override the some of the default page layout properties, such as page orientation.

### **TemplateSrc — Source of template for this reporter**

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

**TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

**LinkTarget — Hyperlink target for this reporter**[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

**Methods**

<code>createTemplate</code>	Create table of contents section template
<code>customizeReporter</code>	Create custom table of contents reporter class
<code>getClassFolder</code>	Table of contents class definition file location
<code>getTitleReporter</code>	Get table of contents title reporter

**Inherited Methods**

<code>copy</code>	Create copy of reporter object and make deep copies of certain property values
<code>getImpl</code>	Get implementation of reporter

**Examples****Default Table of Contents**

Create a table of contents that uses the default formatting.

```
import mlreportgen.report.*
rpt = Report('output', 'pdf');
toc = TableOfContents();
add(rpt, toc);
```

**Customized Table of Contents**

Create a report that includes a table of contents with a title in green. This report also includes chapters, sections, and an appendix section.

```
import mlreportgen.report.*
import mlreportgen.dom.*

rpt = Report('Report with TOC');
add(rpt, TitlePage(Title='Report', Subtitle='with TOC'));
toc = TableOfContents;
toc.Title = Text('Table of Contents');
toc.Title.Color = 'green';
toc.NumberOfLevels = 2;
```



```

add(rpt,toc);

ch = Chapter('First Chapter');
add(ch,Section('First Subsection'));
add(ch,Section('Second Subsection'));

add(rpt,ch);
add(rpt,Chapter('Second Chapter'));

add(rpt,PDFPageLayout);
p = Paragraph('Appendix');
p.Style = {OutlineLevel(1), Bold, FontSize('18pt')};
add(rpt,p);

close(rpt);
rptview(rpt);

```

## Table of Contents

<a href="#">Chapter 1. First Chapter</a> .....	1
<a href="#">1.1. First Subsection</a> .....	1
<a href="#">1.2. Second Subsection</a> .....	1
<a href="#">Chapter 2. Second Chapter</a> .....	2
<a href="#">Appendix</a> .....	3

## Compatibility Considerations

### Use NumberOfLevels and LeaderPattern instead of TOCObj

*Behavior changed in R2021a*

Starting in R2021a, use the `NumberOfLevels` and `LeaderPattern` properties, instead of the `TOCObj` property to specify the number of header levels and leader type. Report generation programs that use the `TOCObj` property will continue to run.

### See Also

`mlreportgen.report.Report` | `mlreportgen.report.Reporter`

### Topics

“What Is a Reporter?” on page 1-3

### Introduced in R2017b

## mlreportgen.report.Title class

**Package:** mlreportgen.report

Fill title hole reporter

### Description

Reporter that other reporters, such as `Chapter`, `FormalImage`, and `BaseTable`, use to fill a title hole in a template.

An instance of this class can be created only by another reporter, which is called the owner. The instance fills a hole in a template that the owner specifies with a title that the owner specifies via an instance template and instance properties that it sets. For example, the `BaseTable` reporter (the owner) uses an instance of this class to fill the `Title` hole in its template with a hierarchically numbered title.

The title generated by an instance of this class consists of static text and dynamic content. The static text is specified by the instance template. The dynamic content is specified by the instance `Content`, `NumberPrefix`, and `NumberSuffix` properties. If the name of a template hole matches one of these property names, this reporter fills the hole with the value property. For example, if the title template contains a `Content` hole and a `NumberPrefix` hole, this reporter fills those holes with the values of the corresponding properties.

The owner specifies the location of the title template via the instance `TemplateSrc` and `TemplateName` properties. The `TemplateSrc` property specifies the source of a template file whose template library contains the title template. The `TemplateName` property specifies the name of the title template. For example, the `BaseTable` reporter template library contains two numbered title templates, one named `BaseTableNumberedTitle` for numbered tables, and one named `BaseTableHierNumberedTitle` for hierarchically numbered tables. The `BaseTable` reporter specifies which template to use by setting this reporter's `TemplateSrc` property to the `BaseTable` template and this reporter's `TemplateName` property to the title template name.

The owner can also generate a hyperlink target in front of the title by setting this reporter's `LinkTarget` property to the id of the link target.

### Properties

#### **HoleId — ID of hole**

string

ID of hole to be filled by this reporter, specified as a string.

#### **Content — Content of hole**

string | character array | ...

Content of hole to be filled by this reporter, specified as one of these values:

- String or character array
- DOM object

- 1-by-*N* or *N*-by-1 array of strings or DOM objects
- 1-by-*N* or *N*-by-1 cell array of strings, character arrays, and/or DOM objects

### **NumberPrefix — Prefix for the title content**

string scalar | character vector

Prefix for the title content, specified as a string scalar or character vector. If no prefix is specified, the default title prefix, translated based on the report locale, is used.

### **NumberSuffix — Suffix for the title content**

string scalar | character vector

Suffix for the title content, specified as a string scalar or character vector. If no suffix is specified, the default title suffix, translated based on the report locale, is used.

### **Translations — Title translations**

structure

Title translations, specified as a MATLAB structure. The structure contains three fields that are empty by default:

- `Translations.NumberPrefixSuffix`

If this field is empty (the default value), the title number prefixes and suffixes are not translated. To generate localized versions of the title number prefix and suffix, the title owner sets this field to a MATLAB `containers.MAP` object that maps a locale key to a structure that contains the translations of the `NumberPrefix` and `NumberSuffix` properties. The value structure must contain these fields:

- `Locale` — ID of the locale, for example, 'en'
- `TitleNumberPrefix` — Locale-specific number prefix, for example, 'Chapter'
- `TitleNumberSuffix` — Locale-specific number suffix, for example, ' . '

- `Translations.Content`

If this field is empty (the default value), the title content is not translated. To generate localized versions of the title content, the title reporter owner sets this field to a MATLAB `containers.Map` object that maps a locale key to the translation of the title content. If the value of this field is not empty and the value of the `Content` property of this reporter is empty, this reporter uses the content map to look up a localized version of the title content.

- `Translations.Owner`

This property is used to find a default title content translation specified by the title owner reporter. This property is for internal use only.

If the specified translation map does not contain a translation for the specified report locale, the en locale is used as the locale for the title, prefix, and suffix. See the `Locale` property of `mlreportgen.report.Report` for information about valid locales.

### **TemplateSrc — Source of template for this reporter**

[] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

**TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template source (`TemplateSrc`) for this reporter.

**LinkTarget — Hyperlink target for this reporter**

[] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

**See Also**

`mlreportgen.report.Report` | `mlreportgen.report.Reporter`

**Topics**

“What Is a Reporter?” on page 1-3

**Introduced in R2018b**

# mlreportgen.report.TitlePage class

**Package:** mlreportgen.report mlreportgen.report mlreportgen.report

**Superclasses:** mlreportgen.report.Reporter

Title page reporter

## Description

Use an object of the mlreportgen.report.TitlePage class to add a title page to a report.

The mlreportgen.report.TitlePage class is a handle class.

## Class Attributes

HandleCompatible	true
------------------	------

## Creation

### Description

tp = mlreportgen.report.TitlePage() creates a title page reporter that uses the default title page template.

tp = mlreportgen.report.TitlePage(Name=Value) sets properties using name-value pairs. You can specify multiple name-value pair arguments in any order.

## Properties

### Title — Title page title

string scalar | character vector | ...

Title page title, specified as one of these values:

- String scalar or character vector
- DOM object
- 1-by-*N* or *N*-by-1 array of strings or DOM objects
- 1-by-*N* or *N*-by-1 cell array of strings, character vectors, or DOM objects
- A Reporter created by the getTitleReporter method of this title page

Example: tp.Title = "My Report";

### Attributes:

GetAccess	public
SetAccess	public

### Subtitle — Report subtitle

string scalar | character vector | ...

Report subtitle, specified as one of these values:

- String scalar or character vector
- DOM object
- 1-by-*N* or *N*-by-1 array of strings or DOM objects
- 1-by-*N* or *N*-by-1 cell array of strings, character vectors, or DOM objects
- Reporter created by the `getSubtitleReporter` method of this title page

Example: `tp.Subtitle = "Part I";`

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**Image — Title page image**

string scalar | character vector | ...

Image to insert in title page, specified as one of these values:

- String scalar or character vector that specifies the file system path of the image
- `mlreportgen.report.Figure` reporter
- DOM object
- 1-by-*N* or *N*-by-1 cell array of image paths, snapshot makers, or DOM objects
- Reporter created by the `getImageReporter` method of this title page

Images formats with these file extensions are supported:

- `.bmp` - Bitmap
- `.gif` - Graphics Interchange Format
- `.jpg` - JPEG
- `.png` - Portable Network Graphics
- `.emf` - Enhanced metafile (supported only in `.docx` output on Windows)
- `.svg` - Scalable Vector Graphic
- `.tif` - Tag Image File

Example: `tp.Image = "reports/imagedir/titleimage.jpg";`

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**Author — Report author**

environment variable user name (default) | string scalar | character vector | ...

Report author, specified as one of these values:

- String scalar or character vector
- DOM object

- 1-by-*N* or *N*-by-1 array of strings or DOM objects
- 1-by-*N* or *N*-by-1 cell array of strings, character vectors, or DOM objects
- Reporter created by the `getAuthorReporter` method of this title page

If the environment variable user name is not found, the default value is empty.

Example: `TitlePage("Author", "John Smith")`

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**Publisher — Report publisher**

`string scalar | character vector | ...`

Report publisher, specified as one of these values:

- String scalar or character vector
- DOM object
- 1-by-*N* or *N*-by-1 array of strings or DOM objects
- 1-by-*N* or *N*-by-1 cell array of strings, character vectors, or DOM objects
- Reporter created by the `getPublisherReporter` method of this title page

Example: `tp.Publisher = "Smith Company";`

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**PubDate — Report publication date**

`string scalar | character vector | ...`

Report publication date, specified as one of these values:

- String scalar or character vector
- DOM object
- 1-by-*N* or *N*-by-1 array of strings or DOM objects
- 1-by-*N* or *N*-by-1 cell array of strings, character vectors, DOM objects
- Reporter created by the `getPubDateReporter` method of this title page

Example: `tp.PubDate = "April 23, 2017";`

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**Layout — Page layout for title page**

`mlreportgen.report.ReporterLayout` object

Page layout for the title page, specified as an `mlreportgen.report.ReporterLayout` object. Use the properties of the `ReporterLayout` object to override the default page layout properties, such as page orientation.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>Restricts access</code>

**TemplateSrc — Source of template for this reporter**

[ ] (default) | character vector | string scalar | reporter or report | DOM document or document part

Source of the template for this reporter, specified in one of these ways:

- Character vector or string scalar that specifies the path of the file that contains the template for this reporter
- Reporter or report whose template is used for this reporter or whose template library contains the template for this reporter
- DOM document or document part whose template is used for this reporter or whose template library contains the template for this reporter

The specified template must be the same type as the report to which this reporter is appended. For example, for a Microsoft Word report, `TemplateSrc` must be a Word reporter template. If the `TemplateSrc` property is empty, this reporter uses the default reporter template for the output type of the report.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**TemplateName — Name of template for this reporter**

character vector | string scalar

Name of the template for this reporter, specified as a character vector or string scalar. The template for this reporter must be in the template library of the template specified by the `TemplateSrc` property of this reporter.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>

**LinkTarget — Hyperlink target for this reporter**

[ ] (default) | character vector | string scalar | `mlreportgen.dom.LinkTarget` object

Hyperlink target for this reporter, specified as a character vector or string scalar that specifies the link target ID, or an `mlreportgen.dom.LinkTarget` object. A character vector or string scalar value is converted to a `LinkTarget` object. The link target immediately precedes the content of this reporter in the output report.

**Attributes:**

<code>GetAccess</code>	<code>public</code>
<code>SetAccess</code>	<code>public</code>



## Methods

### Public Methods

Method	Purpose
mlreportgen.report.TitlePage.createTemplate	Create title page template
mlreportgen.report.TitlePage.customizeReporter	Create custom title page reported class
mlreportgen.report.TitlePage.getClassFolder	Get title page class definition file location
getAuthorReporter	Get title page author reporter
getImageReporter	Get title page image reporter
getPublisherReporter	Get title page publisher reporter
getSubtitleReporter	Get title page subtitle reporter
getTitleReporter	Get title page title reporter
getPubDateReporter	Get title page publication date reporter
getImpl	Get implementation of reporter
copy	Create copy of reporter object and make deep copies of property values that reference a reporter, ReporterLayout, or DOM object

## Examples

### Default Title Page

Create a title page that uses the default formatting. Add the title page to the report and view the report.

```
import mlreportgen.report.*
rpt = Report("output", "pdf");

tp = TitlePage();
tp.Title = "Aircraft Tests";
tp.Subtitle = "Monthly Data";
tp.Image = which("b747.jpg");
tp.Author = "John Smith";
tp.Publisher = "MathWorks";
tp.PubDate = date();

add(rpt, tp);
close(rpt);
rptview(rpt);
```

# Aircraft Tests

## Monthly Data



**John Smith**

MathWorks

19-Jan-2018

### **Title Page with Customized Color**

Create a title page that uses the default title format, but changes the title color to red. Specify the Title property as a DOM Text object and set its color to red.

```

import mlreportgen.report.*
import mlreportgen.dom.*

rpt = Report("output", "pdf");
tp = TitlePage;
tp.Title = Text("Aircraft Tests");
tp.Title.Color = "red";

add(rpt, tp);
close(rpt);
rptview(rpt);

```

### Override Title Page Title Formatting

Create a title page that overrides the title property formatting. Change the title font to 24-point Arial, the title text color to white, and use a blue background. Any styles you do not specify use the `mlreportgen.dom.Paragraph` class defaults.

```

import mlreportgen.report.*
import mlreportgen.dom.*

rpt = Report("output", "pdf");
tp = TitlePage();
title = Paragraph("Aircraft Tests");
title.Style = {HAlign("left"), FontFamily("Arial"), ...
    FontSize("24pt"), Color("white"), ...
    BackgroundColor("blue"), ...
    OuterMargin("0in", "0in", ".5in", "1in"), ...
    HAlign("center")};
tp.Title = title;
tp.Subtitle = "Monthly Data";
tp.Image = which("b747.jpg");
tp.Author = "John Smith";
tp.Publisher = "MathWorks";
tp.PubDate = date();

add(rpt, tp);
close(rpt);
rptview(rpt);

```

### Override Title Page Formatting and Layout Using Custom Template

The template for a `TitlePage` object determines the page orientation, page margins, page size, and other page layout properties. You can customize and override the title page layout by using a customized version of the default template. You can also customize individual title page elements by customizing those element templates. The `TitlePage` reporter supports two approaches to overriding title page element templates.

To use a custom `TitlePage` template:

Create a copy of the default title page template.

Edit the title page element templates as desired in the copy of the template. The names of the templates have the form `TitlePageNAME` where `NAME` is the name of the template in the template library. For example, the name of the title template is `TitlePageTitle`.

Set the `TitlePage TemplateSrc` property of the object to the path of the custom template.

### **Override Title Page Formatting and Layout Using Different template Library**

This approach takes advantage of the fact that the `TitlePage` object uses specialized reporters, called hole reporters, to apply element templates to the elements. Consequently, you can use the `TitlePage` methods for getting the reporter to apply a template to a particular element. For example, the `getTitleReporter` method returns the reporter used for applying the `TitlePageTitle` template to the content of the report title.

Copy the title page element templates that you want to customize into a different template library. For example, you can copy the template library of the report or the template library of a DOM document part object. These template libraries are often libraries that you created to store customized versions of templates.

For each title page element to be customized, get its element reporter. For example, for the title, use the `getTitleReporter` method.

Set the `TemplateSrc` property of the element reporter to the source of the template library containing the customized version of the element template.

Set `Content` property of the element reporter to the element content.

Set the title page object element property to the element reporter object.

```
import mlreportgen.report.*
import mlreportgen.dom.*

rpt = Report("MyReport", "pdf", "MyCustomPDFTemplate");
tp = TitlePage;
titleReporter = getTitleReporter(tp);
titleReporter.TemplateSrc = rpt;
titleReporter.Content = "My Report";
tp.Title = titleReporter;
```

### **See Also**

`mlreportgen.dom.Paragraph`

### **Topics**

“What Is a Reporter?” on page 1-3

### **Introduced in R2017b**

# mlreportgen.utils.HTMLDoc class

**Package:** mlreportgen.utils

Wrap HTML file for viewing

## Description

This utility wraps a .html file into an object. You can then manipulate the object using the methods of this utility.

## Construction

`docobj = mlreportgen.utils.HTMLDoc(filename)` wraps an HTML document file and returns it as an HTMLDoc object. The constructed HTML doc object is not visible. To make it visible, use the `show` method.

## Input Arguments

**filename — Path to HTML file**

string | character vector

Path to an HTML file, specified as a string or character vector.

## Output Arguments

**docobj — HTMLDoc object**

wrapped file

HTMLDoc object, returned as a wrapped document object file.

## Methods

show	Show HTML file <code>mlreportgen.utils.HTMLDoc.show(htmldoc)</code>
hide	Hide HTML file <code>mlreportgen.utils.HTMLDoc.hide(htmldoc)</code>
isVisible	Test whether HTML file is visible <code>mlreportgen.utils.HTMLDoc.isVisible(htmldoc)</code>

## Examples

### Create HTMLDoc Object and Make It Visible

This example wraps the existing `MyHTMLDoc.html` file into an HTMLDoc object.

```
docobj = mlreportgen.utils.HTMLDoc("MyHTMLDoc.html");  
show(docobj);
```

### **See Also**

`mlreportgen.utils.HTMXDoc` | `mlreportgen.utils.tidy`

**Introduced in R2018b**

# mlreportgen.utils.HTMXDoc class

**Package:** mlreportgen.utils

Wrap HTMX document file for viewing

## Construction

`docobj = mlreportgen.utils.HTMXDoc(filename)` wraps an HTMX document file and returns it as an HTMX doc object. The HTMX doc object is not visible on construction. To make it visible, use the `show` method.

## Input Arguments

**filename — Path to HTMX file**

string | character vector

Path to HTMX file, specified as a string or character vector.

## Output Arguments

**docobj — HTMX document object**

wrapped file

HTMX document object, returned as a wrapped document object file.

## Properties

**FileName — Full path to HTMX file**

string | character vector

Full path to HTMX file, specified as a string or character vector.

## Methods

<code>show</code>	Show HTMX file <code>mlreportgen.utils.HTMXDoc.show(htmldoc)</code>
<code>hide</code>	Hide HTMX file <code>mlreportgen.utils.HTMXDoc.hide(htmldoc)</code>
<code>isVisible</code>	Test whether HTMX file is visible <code>mlreportgen.utils.HTMXDoc.isVisible(htmldoc)</code>

## Examples

### Create HTMX Doc Object and Make It Visible

This example wraps the existing `MyHTMXDoc.html` file into an `HTMXDoc` object.

```
docobj = mlreportgen.utils.HTMXDoc("MyHTMXDoc.htmx");  
show(docobj);
```

### See Also

`mlreportgen.utils.HTMLDoc` | `mlreportgen.utils.tidy`

**Introduced in R2018b**



# mlreportgen.utils.PDFDoc class

**Package:** mlreportgen.utils

Wrap PDF file for viewing

## Construction

`docobj = mlreportgen.utils.PDFDoc(filename)` wraps a PDF document file for viewing. The PDF document object is not visible on construction. To make it visible, use the `show` method.

## Input Arguments

**filename** — Path to PDF file

string | character vector

Path to a PDF file, specified as a string or character vector.

## Output Arguments

**docobj** — PDF document object

wrapped file

PDF document object, returned as a wrapped document object file.

## Properties

**FileName** — Full path to PDF file

string | character vector

Full path to PDF file, specified as a string or character vector.

## Methods

show	Show PDF file <code>mlreportgen.utils.PDFDoc.show(pdfDoc)</code>
hide	Hide PDF file <code>mlreportgen.utils.PDFDoc.hide(pdfDoc)</code>
isVisible	Check whether PDF file is visible <code>mlreportgen.utils.PDFDoc.isVisible(pdfDoc)</code>

## Examples

### Create PDF Doc Object and Make It Visible

```
docobj = mlreportgen.utils.PDFDoc("MyPDFDoc.pdf");
show(docobj);
```

**See Also**

`mlreportgen.utils.rptviewer`

**Introduced in R2018b**

# mlreportgen.utils.powerpoint class

**Package:** mlreportgen.utils

Interact with PowerPoint application and presentations

## Description

This utility provides methods for interacting with the PowerPoint application and PowerPoint presentations. These methods use the MATLAB interface to the Microsoft .NET framework to interact with the PowerPoint editor.

## Methods

- “start” on page 12-907
- “load” on page 12-907
- “open” on page 12-907
- “close” on page 12-908
- “closeAll” on page 12-908
- “show” on page 12-909
- “hide” on page 12-909
- “isAvailable” on page 12-909
- “isAvailable” on page 12-909
- “isStarted” on page 12-909
- “isLoading” on page 12-909
- “pptapp” on page 12-909
- “pptpres” on page 12-910

### start

```
pptApp = mlreportgen.utils.powerpoint.start()
```

Start the PowerPoint application if it has not already been started and return a pptApp object. A pptApp object is a MATLAB object that wraps the .NET object that wraps the PowerPoint editor. The PowerPoint application is hidden.

### load

```
pptPres = mlreportgen.utils.powerpoint.load(filename)
```

Load an existing PowerPoint presentation file specified in filename and return a pptPres object. A pptPres object is a MATLAB object that wraps the .NET object that wraps the PowerPoint presentation.

### open

```
pptPres = mlreportgen.utils.powerpoint.open(filename)
```

Open a PowerPoint presentation file specified in `filename`, make it visible, and return a `pptPres` object. A `pptPres` object is a MATLAB object that wraps the .NET object that wraps the PowerPoint presentation.

**close**

```
tf = mlreportgen.utils.powerpoint.close
tf = mlreportgen.utils.powerpoint.close(true)
tf = mlreportgen.utils.powerpoint.close(false)

tf = mlreportgen.utils.powerpoint.close(filename)
tf = mlreportgen.utils.powerpoint.close(filename,true)
tf = mlreportgen.utils.powerpoint.close(filename,false)
```

**Close PowerPoint Application**

For empty or no input, close the PowerPoint application only if there are no unsaved PowerPoint presentations.

For `true` input, close the PowerPoint application only if there are no unsaved PowerPoint presentations.

For `false` input, close the PowerPoint application even if there are unsaved PowerPoint presentations or if there are PowerPoint presentations opened outside of MATLAB.

For each of these syntaxes, return `true` if application is closed or `false` if application remains open.

**Close PowerPoint Presentation**

For `filename` input, close the PowerPoint presentation specified as `filename` only if there are no unsaved changes.

For `filename` and `true` inputs, close the PowerPoint presentation, specified as `filename`, only if there are no unsaved changes.

For `filename` and `false` inputs, close the PowerPoint presentation, specified as `filename`, even if there are unsaved changes.

For each of these syntaxes,

- Hide the PowerPoint application if there are no other open PowerPoint presentations.
- Return `true` if the PowerPoint presentation file is closed
- Return `false` if the PowerPoint presentation file remains open

**closeAll**

```
tf = mlreportgen.utils.powerpoint.closeAll()
tf = mlreportgen.utils.powerpoint.closeAll(true)
tf = mlreportgen.utils.powerpoint.closeAll(false)
```

For empty input, close all PowerPoint presentation files and hide the PowerPoint application.

For `true` input, close all PowerPoint presentation only if there are no unsaved changes.

For `false` input, close all PowerPoint presentation files even if there are unsaved changes.

For each of these syntaxes:

- Hide the PowerPoint application if there are no other open PowerPoint presentation files
- Return `true` if all PowerPoint presentation files are closed
- Return `false` if any PowerPoint presentation file remains open

**show**

```
pptApp = mlreportgen.utils.powerpoint.show()  
pptPres = mlreportgen.utils.powerpoint.show(filename)
```

For empty input, make the PowerPoint application or PowerPoint presentation file visible and return the `pptApp` object.

For `filename` input, make the specified PowerPoint presentation file visible and returns the `pptPres` object.

**hide**

```
pptApp = mlreportgen.utils.powerpoint.hide()  
pptPres = mlreportgen.utils.powerpoint.hide(filename)
```

For empty input, hide the PowerPoint application or PowerPoint presentation and returns the `pptApp` object.

For `filename` input, hide the specified PowerPoint presentation file and returns the `pptPres` object.

**isAvailable**

```
files = mlreportgen.utils.powerpoint.fileNames()
```

Return a string array of file names for open PowerPoint presentation files.

**isAvailable**

```
tf = mlreportgen.utils.powerpoint.isAvailable()
```

Verify whether PowerPoint is available for use. Return `true` if PowerPoint is available or `false` if it is not available.

**isStarted**

```
tf = mlreportgen.utils.powerpoint.isStarted
```

Verify whether PowerPoint application is started. Return `true` if PowerPoint is started or `false` if it is not started.

**isLoaded**

```
tf = mlreportgen.utils.powerpoint.isLoaded(filename)
```

Verify whether PowerPoint presentation file is loaded. Return `true` if PowerPoint presentation `filename` is loaded or `false` if it is not loaded.

**pptapp**

```
pptApp = mlreportgen.utils.powerpoint.pptApp()
```

Return `pptApp` object. An error occurs if PowerPoint is not started.

**pptpres**

```
pptPres = mlreportgen.utils.powerpoint.pptPres(filename)
```

Return pptPres object that wraps the PowerPoint presentation file specified in filename. An error occurs if the PowerPoint presentation file does not exist.

**Examples****Open PowerPoint Presentations**

Open the test.pptx and test1.pptx PowerPoint presentations, which are in the current working folder.

```
pptPres = mlreportgen.utils.powerpoint.open('test')
pptPres1 = mlreportgen.utils.powerpoint.open('test1')
```

```
pptPres =
```

```
    PPTPres with properties:
        FileName: 'C:\Users\username\Documents\test.pptx'
```

```
pptPres1 =
```

```
    pptPres with properties:
        FileName: 'C:\Users\username\Documents\test1.pptx'
```

**Obtain PowerPoint Presentation File Names**

Obtain the names of open PowerPoint presentation files.

```
files = mlreportgen.utils.powerpoint.fileNames()
```

```
files =
```

```
    1x2 string array

    "C:\Users\username\Documents\test.pptx" ...
    "C:\Users\username\Documents\test1.pptx"
```

**See Also**

[mlreportgen.utils.PPTPres](#) | [mlreportgen.utils.PPTApp](#)

**Topics**

“Getting Started with Microsoft .NET”

**Introduced in R2018b**

## mlreportgen.utils.PPTApp class

**Package:** mlreportgen.utils

Wrap PowerPoint application

### Construction

mlreportgen.utils.PPTApp wraps a PowerPoint application as a .NET object. Only one PowerPoint application object can be active to use this utility. To obtain the active PowerPoint application object, use the `instance` method.

### Methods

instance	Return active PowerPoint application object  pptapp = mlreportgen.utils.PPTApp.instance()
show	Show PowerPoint application  pptapp = show(pptapp)
hide	Hide PowerPoint application  pptapp = hide(pptapp)
close	Close PowerPoint application  pptapp = close(pptapp)
isOpen	Test if PowerPoint application is open  pptapp = isOpen(pptapp)
isVisible	Test if PowerPoint application is visible  pptapp = isVisible(pptapp)
netobj	Return a .NET PowerPoint application object  netpptappobj = netobj(pptapp)

### Examples

#### Find and Hide PowerPoint Application Instance

```
pptapp = mlreportgen.utils.PPTApp.instance();
hide(pptapp)
```

#### See Also

mlreportgen.utils.PPTPres

#### External Websites

<https://docs.microsoft.com/en-us/office/vba/api/overview/powerpoint>

**Introduced in R2018b**



# mlreportgen.utils.PPTPres class

**Package:** mlreportgen.utils

Wrap PowerPoint presentation file

## Construction

`pptwrap = mlreportgen.utils.PPTPres(filename)` wraps a PowerPoint presentation file, specified as `filename`. There can be only one PPTPres object for a `filename`.

## Input Arguments

**filename** — Path to PowerPoint presentation file

string | character vector

See FileName property.

## Output Arguments

**pptwrap** — Wrapped PowerPoint presentation file

character vector

Wrapped PowerPoint presentation file, returned as a character vector.

## Properties

**FileName** — Full path to PowerPoint presentation file

string | character vector

Full path to PowerPoint presentation file, specified as a string or character vector.

## Methods

show	Make PowerPoint presentation visible <code>show(pptPres)</code>
hide	Hide PowerPoint presentation <code>hide(pptPres)</code>
close	Close PowerPoint presentation <code>close(pptPres)</code>
save	Save PowerPoint presentation <code>save(pptPres)</code>
print	Print PowerPoint presentation <code>print(pptPres)</code>

<code>exportToPDF</code>	Export to PDF document with same file name <code>exportToPDF(pptPres)</code> Export to PDF document with specified file name <code>pdfFullPath = exportToPDF(pptPres, pdfFileName)</code>
<code>isOpen</code>	Test if PowerPoint presentation is open <code>isOpen(pptPres)</code>
<code>isReadOnly</code>	Test if PowerPoint presentation is read-only <code>isReadOnly(pptPres)</code>
<code>isSaved</code>	Test if PowerPoint presentation is saved <code>isSaved(pptPres)</code>
<code>isVisible</code>	Test if PowerPoint presentation is visible <code>isVisible(pptPres)</code>
<code>netobj</code>	Return a .NET Word document object, which allows using the .NET interface on this object <code>netobj(pptPres)</code>

## Examples

### Create PowerPoint Presentation and Test Whether It Is Open

This example creates a PowerPoint presentation object from the existing `MyPPTDoc.ppt` file.

```
pptwrap = mlreportgen.utils.PPTPres("MyPPTDoc.ppt");
isOpen(pptwrap)
```

### See Also

`mlreportgen.utils.powerpoint` | `mlreportgen.utils.PPTApp`

### External Websites

<https://docs.microsoft.com/en-us/office/vba/api/overview/powerpoint>

**Introduced in R2018b**

## mlreportgen.utils.rptviewer class

**Package:** mlreportgen.utils

Report document viewer manager

### Construction

mlreportgen.utils.rptviewer manages report document viewers.

### Methods

open	Open the appropriate viewer for filename. mlreportgen.utils.rptviewer.open(filename)
close	Close the viewer associated with filename . mlreportgen.utils.rptviewer.close(filename)
isOpen	Test whether filename is assigned to a viewer. mlreportgen.utils.rptviewer.isOpen(filename)
closeAll	Close all open viewers. mlreportgen.utils.rptviewer.closeAll()

### Examples

#### Open File in Appropriate Viewer

```
mlreportgen.utils.rptviewer('mydoc.pdf')
```

#### See Also

mlreportgen.utils.word | mlreportgen.utils.HTMXDoc | mlreportgen.utils.HTMLDoc | mlreportgen.utils.PDFDoc | mlreportgen.utils.PPTPres | mlreportgen.utils.WordDoc | mlreportgen.utils.powerpoint

**Introduced in R2018b**

## mlreportgen.utils.TableSlice class

**Package:** mlreportgen.utils

Table slicer object output

### Description

Contains a slice of a table generated by an mlreportgen.utils.TableSlicer object. You do not need to create instances of this mlreportgen.utils.TableSlice class.

### Properties

#### **Table** — Slice of a table

array of columns

This property is read-only. Its value is a table generated by an mlreportgen.utils.TableSlicer object.

#### **StartCol** — Starting column of table slice

positive integer

This property is read-only. Its value is the index of the column from the original table where this slice starts.

#### **EndCol** — Ending column of slice

positive integer

This property is read-only. Its value is the index of the column from the original table where this slice ends.

### See Also

mlreportgen.utils.TableSlicer

**Introduced in R2018b**

# mlreportgen.utils.TableSlicer class

**Package:** mlreportgen.utils

Divide table into slices

## Description

Divides a table vertically into a set of narrower tables (slices). To divide a table that is too wide to fit legibly on a page into a set of legible slices, use this `TableSlicer` object.

## Construction

`slicer = mlreportgen.utils.TableSlicer()` creates an empty table slicer object. Use its properties to specify the input table to slice, the maximum number of columns per slice, and the number of columns to repeat.

---

**Note** To slice a table generated by the `mlreportgen.report.BaseTable` reporter, set the `MaxCols` property of the `BaseTable` reporter to the size of the slices you want to generate. You do not need to use this `TableSlicer` utility to set the slice width.

---

`slicer = mlreportgen.utils.TableSlicer(Name=Value)` creates a table slicer object with additional options specified by one or more `Name=Value` pair arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1=Value1, ..., NameN=ValueN`.

## Properties

### Table — Input table object to be sliced

mlreportgen.dom.Table object | mlreportgen.dom.FormalTable object

Input table object to be sliced, specified as a DOM Table object or Formal Table object. For both DOM Table and Formal Table inputs, the table must have the same number of columns in each row. Its `RowSpan` and `ColSpan` values must be empty [] or 1. If a Formal Table object has headers or footers, the number of header or footer columns must match the number of columns in the table body.

### MaxCols — Maximum number of columns to display per table slice

Inf (default) | positive integer

Maximum number of columns to display per table slice, specified as `Inf` or as a positive integer. If the value of this property is `Inf`, all original table columns are included in a single table. A `MaxCols` value greater than or equal to the number of table columns also produces a single table with all columns. Large table data sets can cause illegible tables to be generated. Set this property to the number of columns from the original table that fit legibly on a page. To determine an optimal value, iterate setting the `MaxCols` value and viewing the report.

### RepeatCols — Number of initial columns to repeat per slice

0 (default) | positive integer

Number of initial columns to repeat per slice, specified as 0 or a positive integer. A nonzero number, *n*, repeats the first *n* columns of the original table in each slice. The `MaxCols` property value includes the `RepeatCols` property value. For example, if `MaxCols` is 6 and `RepeatCols` is 2, each table slice has a total of six columns with the first two columns repeated from the original table.

## Methods

`slices = slice(slicer)` returns an array of `mlreportgen.utils.TableSlice` objects, each containing a table slice, and the start and end column indices of the slice. The start and end column indices refer to the column indices of the original input table.

## Examples

### Slice a Formal Table

Create a `FormalTable` object that contains employee data. Slice the table so that the first table column repeats in each slice and the maximum number of columns in each slice is three.

```
employee_data = {...
    'Joe Smith', '3/12/06', 'Engineer', 'A302';...
    'Mary Jones', '4/17/03', 'Writer', 'C312';...
    'John Johnson', '9/5/12', 'Sr. Programmer', 'A421';...
    'Susan White', '6/29/16', 'Sr. Engineer', 'B201';...
    'Thomas Lee', '10/1/17', 'QE Engineer', 'C200'};
tbl_header = {'Name', 'Hire Date', 'Position', 'Office'};

import mlreportgen.report.*
import mlreportgen.dom.*
import mlreportgen.utils.*

rpt = mlreportgen.report.Report("Sliced Table", 'pdf');
open(rpt);

chapter = Chapter("Title", 'Employee Report');
table = FormalTable(tbl_header, employee_data);
table.Border = 'Solid';
table.RowSep = 'Solid';
table.ColSep = 'Solid';

para = Paragraph(['The table is sliced into two tables, '...
    'with the first column repeating in each table.']);
para.Style = {OuterMargin('0in', '0in', '0in', '12pt')};
para.FontSize = '14pt';
add(chapter, para)

slicer = TableSlicer("Table", table, "MaxCols", 3, "RepeatCols", 1);
totcols = slicer.MaxCols - slicer.RepeatCols;
slices = slicer.slice();
for slice=slices
    str = sprintf('%d repeating column and up to %d more columns',...
        slicer.RepeatCols, totcols);
    para = Paragraph(str);
    para.Bold = true;
```

```

    add(chapter, para)
    add(chapter, slice.Table)
end

add(rpt, chapter)
close(rpt)
rptview(rpt)

```

## Chapter 1. Employee Report

The table is sliced into two tables, with the first column repeating in each table.

**1 repeating column and up to 2 more columns**

Name	Hire Date	Position
Joe Smith	3/12/06	Engineer
Mary Jones	4/17/03	Writer
John Johnson	9/5/12	Sr. Programmer
Susan White	6/29/16	Sr. Engineer
Thomas Lee	10/1/17	QE Engineer

**1 repeating column and up to 2 more columns**

Name	Office
Joe Smith	A302
Mary Jones	C312
John Johnson	A421
Susan White	B201
Thomas Lee	C200

### See Also

mlreportgen.utils.TableSlice | mlreportgen.dom.Table | mlreportgen.dom.FormalTable

**Introduced in R2018b**

## mlreportgen.utils.units class

**Package:** mlreportgen.utils

Convert units

### Description

The units class provides conversions between units.

### Construction

`value = mlreportgen.utils.units.toPixels(lengthUnits)` converts the specified `lengthUnits` to pixels. The `lengthUnits` has two parts. The length is the number of units, such as 5 or 8.3. The unit is the unit type, such as `pixels` or `inches`. You can also use a syntax with separate length and units: `value = mlreportgen.utils.units.toPixels(length,units)`.

`value = mlreportgen.utils.units.toPoints(lengthUnits)` converts the specified `lengthUnits` to points. You can also use a syntax with separate length and units: `value = mlreportgen.utils.units.toPoints(length,units)`.

`value = mlreportgen.utils.units.toPoints... (length,units,"DPI",convFactor)` uses the optional Name-Value pair to override the screen pixels to inches. This syntax can be used with any conversion syntax (that is, `toPixels`, `toPoints`, etc.)

`value = mlreportgen.utils.units.toInches(lengthUnits)` converts the specified `lengthUnits` to inches. You can also use a syntax with separate length and units: `value = mlreportgen.utils.units.toInches(length,units)`.

`value = mlreportgen.utils.units.toCentimeters(lengthUnits)` converts the specified `lengthUnits` to centimeters. You can also use a syntax with separate length and units: `value = mlreportgen.utils.units.toCentimeters(length,units)`.

`value = mlreportgen.utils.units.toMillimeters(lengthUnits)` converts the specified `lengthUnits` to millimeters. You can also use a syntax with separate length and units: `value = mlreportgen.utils.units.toMillimeters(length,units)`.

`value = mlreportgen.utils.units.toPicas(lengthUnits)` converts the specified `lengthUnits` to picas. You can also use a syntax with separate length and units: `value = mlreportgen.utils.units.toPicas(length,units)`.

### Input Arguments

#### **lengthUnits** — Measurement length and units to convert

string | character vector

Measurement length and units to convert, specified as a string.

Example: "5in"

#### **length** — Length to convert

positive numeric



Length to convert, specified as a positive numeric value.

Example: 5 (as in (5, "in"))

### **units** – Unit type to convert

pixels | centimeters | inches | millimeters | picas | points

Unit type to convert, specified as singular or plural units or a units abbreviation.

<b>Units (singular)</b>	<b>Units (plural)</b>	<b>Abbreviation</b>
inch	inches	in
centimeter	centimeters	cm
millimeter	millimeters	mm
pixel	pixels	px
pica	picas	pc
point	points	pt

Example: "5in", "5inch", or "5inches"

### **convFactor** – Conversion factor

positive numeric

Conversion factor for overriding screen pixels to inches, specified as a positive numeric value. Use a Name-Value pair, where "DPI" is the Name character vector or string and `dpi` is the number of dots per inch. This input is optional.

### **Outputs**

#### **value** – Value in converted units

positive numeric

Value in converted units, returned as the units specified by the method name.

## **Examples**

### **Convert Units**

```
value = mlreportgen.utils.units.toPixels("5in");
value = mlreportgen.utils.units.toPoints("8.3cm");
value = mlreportgen.utils.units.toMillimeters(3,"in");
value = mlreportgen.utils.units.toPicas(7,"pt");
value = mlreportgen.utils.units.toInches(3,"in",...
    "DPI",96);
```

### **Introduced in R2018b**

## mlreportgen.utils.word class

**Package:** mlreportgen.utils mlreportgen.utils mlreportgen.utils  
 mlreportgen.utils mlreportgen.utils mlreportgen.utils mlreportgen.utils  
 mlreportgen.utils mlreportgen.utils mlreportgen.utils mlreportgen.utils  
 mlreportgen.utils mlreportgen.utils mlreportgen.utils

Interact with Microsoft Word and Word documents

### Description

Use the static methods of the `mlreportgen.utils.word` class to interact with the Microsoft Word application and Word documents. These methods use the MATLAB interface to the Microsoft .NET Framework to interact with Word.

The `mlreportgen.utils.word` class is a handle class.

### Class Attributes

`HandleCompatible` true

For information on class attributes, see “Class Attributes”.

### Methods

#### Public Methods

<code>mlreportgen.utils.word.start</code>	<p><code>wordApp = mlreportgen.utils.word.start()</code> starts the Word application if it has not already been started and returns an <code>mlreportgen.utils.WordApp</code> object. A <code>WordApp</code> object wraps a .NET object that wraps the Word editor. The Word application is hidden.</p>
<code>mlreportgen.utils.word.load</code>	<p><code>wordDoc = mlreportgen.utils.word.load(filename)</code> loads the existing Word document file specified by <code>filename</code> and returns a <code>mlreportgen.utils.WordDoc</code> object. A <code>WordDoc</code> object wraps a .NET object that wraps the Word document.</p> <p>Specify <code>filename</code> as a string scalar or character vector.</p>

mlreportgen.utils.word.open	<p>wordDoc = mlreportgen.utils.word.open(filename) loads the Word document specified by filename, makes it visible, and returns a mlreportgen.utils.WordDoc object.</p> <p>A WordDoc object wraps a .NET object that wraps the Word document.</p> <p>Specify filename as a string scalar or character vector.</p>
-----------------------------	---

<code>mlreportgen.utils.word.close</code>	<p>These syntaxes close the Word application and return <code>true</code> if the application is closed and <code>false</code> if the application remains open:</p> <ul style="list-style-type: none"><li>• <code>tf = mlreportgen.utils.word.close()</code> closes the Word application only if there are no unsaved Word documents.</li><li>• <code>tf = mlreportgen.utils.word.close(true)</code> closes the Word application only if there are no unsaved Word documents or there are no Word documents open outside of MATLAB.</li><li>• <code>tf = mlreportgen.utils.word.close(false)</code> closes the Word application even if there are unsaved Word documents or Word documents opened outside of MATLAB.</li></ul> <p>These syntaxes close the Word document specified by <code>filename</code>, return <code>true</code> if the Word document is closed, return <code>false</code> if the Word document remains open, and hide the Word application if there are no open Word documents:</p> <ul style="list-style-type: none"><li>• <code>tf = mlreportgen.utils.word.close(filename)</code> closes the specified Word document file only if there are no unsaved changes.</li><li>• <code>tf = mlreportgen.utils.word.close(filename,true)</code> closes the specified Word document file only if there are no unsaved changes.</li><li>• <code>tf = mlreportgen.utils.word.close(filename,false)</code> closes the specified Word document file even if there are unsaved changes.</li></ul> <p>Specify <code>filename</code> as a string scalar or character vector.</p>
---	--

<code>mlreportgen.utils.word.closeAll</code>	<p>These syntaxes close all open Word documents, return <code>true</code> if all Word documents are closed, return <code>false</code> if a Word document remains open, and hide the Word application if there are no open Word documents:</p> <ul style="list-style-type: none"> <li>• <code>tf = mlreportgen.utils.word.closeAll()</code> closes all Word document files.</li> <li>• <code>tf = mlreportgen.utils.word.closeAll(true)</code> closes all Word document files only if there are no unsaved changes.</li> <li>• <code>tf = mlreportgen.utils.word.closeAll(false)</code> closes all Word document files even if there are no unsaved changes.</li> </ul> <p>Specify filename as a string scalar or character vector.</p>
<code>mlreportgen.utils.word.show</code>	<p><code>wordApp = mlreportgen.utils.word.show()</code> makes the Word application visible and returns the associated <code>mlreportgen.utils.WordApp</code> object.</p> <p><code>wordApp = mlreportgen.utils.word.show(filename)</code> makes the Word document file specified by <code>filename</code> visible and returns the associated <code>mlreportgen.utils.WordDoc</code> object.</p> <p>Specify filename as a string scalar or character vector.</p>
<code>mlreportgen.utils.word.hide</code>	<p><code>wordApp = mlreportgen.utils.word.hide()</code> hides the Word application and returns the <code>mlreportgen.utils.WordApp</code> object associated with the application.</p> <p><code>wordApp = mlreportgen.utils.word.hide(filename)</code> hides the Word document file specified by <code>filename</code> and returns the associated <code>mlreportgen.utils.WordDoc</code> object.</p> <p>Specify filename as a string scalar or character vector.</p>

<code>mreportgen.utils.word.fileNames</code>	<code>files = mreportgen.utils.word.fileNames()</code> returns the open Word document file names as a string array.
<code>mreportgen.utils.word.isAvailable</code>	<code>tf = mreportgen.utils.word.isAvailable()</code> returns true if Word is available for use and false if Word is not available for use.
<code>mreportgen.utils.word.isStarted</code>	<code>tf = mreportgen.utils.word.isStarted()</code> returns true if the Word application is started and false if the Word application is not started.
<code>mreportgen.utils.word.isLoaded</code>	<code>tf = mreportgen.utils.word.isLoaded(filename)</code> returns true if the Word document specified by filename is loaded and false if it is not loaded.  Specify filename as a string scalar or character vector.
<code>mreportgen.utils.word.wordApp</code>	<code>wordApp = mreportgen.utils.word.wordApp()</code> returns the <code>mreportgen.utils.WordApp</code> object associated with the Word application and an error if the application is not started.
<code>mreportgen.utils.word.wordDoc</code>	<code>wordDoc = mreportgen.utils.word.wordDoc(filename)</code> returns the <code>mreportgen.utils.WordDoc</code> object associated with the Word document file specified by filename and an error if the Word document file is not open.  Specify filename as a string scalar or character vector.

## Examples

### Open Word Documents

Suppose that the Word files, `test.docx` and `test1.docx` are in a folder named `mydocs` on the `C:` drive. Open the files using `mreportgen.utils.word.open`.

```
wordDoc = mreportgen.utils.word.open("test")
wordDoc1 = mreportgen.utils.word.open("test1")
```

```
wordDoc =
```

```
WordDoc with properties:
```

```
FileName: "C:\mydocs\test.docx"
```

```
wordDoc1 =  
    WordDoc with properties:  
        FileName: "C:\mydocs\test1.docx"
```

### Obtain Document File Names

Suppose that the Word files `test.docx` and `test1.docx` are in a folder named `mydocs` on the `C:` drive and that you opened them using `mlreportgen.utils.word.open`.

Obtain the file names of the open Word document files by using `mlreportgen.utils.word.fileNames`.

```
files = mlreportgen.utils.word.fileNames()  
files =  
    1x2 string array  
        "C:\mydocs\test.docx"    "C:\mydocs\test1.docx"
```

### See Also

`mlreportgen.utils.WordDoc` | `mlreportgen.utils.WordApp`

### Topics

“Getting Started with Microsoft .NET”

### Introduced in R2018b

## mlreportgen.utils.WordApp class

**Package:** mlreportgen.utils

Wrap Microsoft Word application

### Construction

mlreportgen.utils.WordApp wraps a Word application as a .NET object. Only one Word application object can be active to use this utility. To obtain the active Word application object, use the instance method.

### Methods

instance	Return active Word application object  wdapp = mlreportgen.utils.WordApp.instance()
show	Show Word application  show(wdapp)
hide	Hide Word application  hide(wdapp)
close	Close Word application  close(wdapp)
isOpen	Test if Word application is open  isOpen(wdapp)
isVisible	Test if Word application is visible  isVisible(wdapp)
netobj	Return a .NET Word application object  netwdappobj = netobj(wdapp)

### Examples

#### Find and Close Word Application Instance

```
wdapp = mlreportgen.utils.WordApp.instance();
close(wdapp)
```

#### See Also

mlreportgen.utils.word | mlreportgen.utils.WordDoc

#### External Websites

<https://docs.microsoft.com/en-us/office/vba/api/overview/word>



**Introduced in R2018b**

## mlreportgen.utils.WordDoc class

**Package:** mlreportgen.utils

Wrap a Microsoft Word doc file

### Description

This utility wraps a Word document file (.docx or .rtf) into an object. You can then manipulate the object using the methods of this utility.

### Construction

`docobj = mlreportgen.utils.WordDoc(filename)` wraps a Word document file and returns it as a Word doc object. Only one WordDoc object can exist for each Word document file.

### Input Arguments

**filename — Path to Word document file**

string | character vector

Path to a Word document file, specified as a string or character vector.

### Output Arguments

**docobj — Word doc object**

wrapped file

WordDoc object, returned as a wrapped document object file.

### Properties

**FileName — Path to Word document file**

string | character vector

Path to a Word document file, specified as a string or character vector.

### Methods

show	Make Word document visible <code>show(wordDoc)</code>
hide	Hide Word document <code>hide(wordDoc)</code>

close	<p>Close Word document</p> <p><code>close(wordDoc)</code> or <code>close(wordDoc, true)</code> closes the Word document only if there are no unsaved changes.</p> <p><code>close(wordDoc, false)</code> closes the Word document even if there are unsaved changes.</p>
save	<p>Save Word document</p> <p><code>save(wordDoc)</code></p>
update	<p>Update Word document fields</p> <p><code>update(wordDoc)</code></p> <p><code>update(wordDoc, 0)</code> forces update even if there are unsaved document fields</p>
print	<p>Print Word document</p> <p><code>print(wordDoc)</code> prints the specified Word document and does not scale the contents to fit to A4 or 8.5-by-11 paper sizes.</p> <p><code>print(wordDoc, "ScaleToFitPaper", true)</code> prints the specified Word document and scales the contents to fit A4 or 8.5-by-11 paper sizes.</p> <p><code>print(wordDoc, "ScaleToFitPaper", false)</code> prints the specified Word document and does not scale the contents to fit to A4 or 8.5-by-11 paper sizes.</p>
saveAsDoc	<p>Save as doc file with same file name</p> <p><code>saveAsDoc(wordDoc)</code></p> <p>Save as doc file with specified file name</p> <p><code>docFullPath = saveAsDoc(wordDoc, docFileName)</code></p>
exportToPDF	<p>Export to PDF document with same file name</p> <p><code>exportToPDF(wordDoc)</code></p> <p>Export to PDF document with specified file name</p> <p><code>pdfFullPath = exportToPDF(wordDoc, pdfFileName)</code></p>

<code>unlinkFields</code>	<p>Remove links from fields in the Word document</p> <p><code>unlinkFields(wordDoc)</code> removes links from all fields</p> <p><code>unlinkFields(wordDoc, fieldType1)</code> removes links from all instances of <code>fieldType1</code> fields. For example, <code>unlinkFields(wordDoc, 'wdHyperLink')</code> removes all hyperlink fields.</p> <p><code>unlinkFields(wordDoc, fieldType1, fieldType2)</code> removes links from all instances of both <code>fieldType1</code> and <code>fieldType2</code> fields.</p> <p>See Word FieldType Enumeration for information on Word fields.</p>
<code>unlinkSubdocuments</code>	<p>Remove links to subdocuments and copy subdocuments into master document</p> <p><code>unlinkSubdocuments(wordDoc)</code></p>
<code>isOpen</code>	<p>Test if Word document is open</p> <p><code>isOpen(wordDoc)</code></p>
<code>isReadOnly</code>	<p>Test if Word document is read-only</p> <p><code>isReadOnly(wordDoc)</code></p>
<code>isSaved</code>	<p>Test if Word document is saved</p> <p><code>isSaved(wordDoc)</code></p>
<code>isVisible</code>	<p>Test if Word document is visible</p> <p><code>isVisible(wordDoc)</code></p>
<code>netobj</code>	<p>Return a .NET Word document object, which allows using the .NET interface on this object</p> <p><code>netobj(wordDoc)</code></p>

## Examples

### Create WordDoc Object and .NET Object

This example creates a WordDoc object from the existing MyWordDoc.docx file.

```
docobj = mlreportgen.utils.WordDoc("MyWordDoc.docx");
netobj(docobj);
```

### See Also

`mlreportgen.utils.word` | `mlreportgen.utils.WordApp`

### External Websites

<https://msdn.microsoft.com/en-us/library/microsoft.office.interop.word.document.aspx>

### Introduced in R2018b

# Create a Report Program

---

- “Create a Report Program” on page 13-3
- “Construct a Report API or DOM API Object” on page 13-6
- “Import the API Packages” on page 13-7
- “Create Report Container” on page 13-8
- “Add Content to a Report” on page 13-10
- “Add Content as a Group” on page 13-12
- “Output Types and Report Generator Packages” on page 13-14
- “Close a Report” on page 13-15
- “Display Report” on page 13-16
- “Report Formatting Approaches” on page 13-17
- “Use Style Sheet Styles” on page 13-19
- “Format Inheritance” on page 13-21
- “Templates for DOM API Report Programs” on page 13-22
- “Form-Based Reporting” on page 13-24
- “Fill the Blanks in a Report Form” on page 13-25
- “Use Subforms in a Report” on page 13-26
- “Create a Microsoft Word Document Part Template Library” on page 13-27
- “Create an HTML Document Part Template Library” on page 13-30
- “Create a PDF Document Part Template Library” on page 13-32
- “Simplify Filling in Forms” on page 13-37
- “Create and Format Text” on page 13-39
- “Format Numbers” on page 13-43
- “Create and Format Paragraphs” on page 13-46
- “Create and Format Lists” on page 13-51
- “Choose Type of Table to Create” on page 13-62
- “Create Informal Tables” on page 13-64
- “Create Formal Tables” on page 13-66
- “Create Tables from MATLAB Tables” on page 13-68
- “Format Tables” on page 13-72
- “Create Links” on page 13-82
- “Create a Dynamic Table” on page 13-85
- “Create and Format Images” on page 13-88
- “Create a Title Page” on page 13-90
- “Create a Table of Contents” on page 13-92
- “Create Image Maps” on page 13-99

- “Automatically Number Document Content” on page 13-101
- “Convert HTML Content to a DOM Object” on page 13-105
- “Prepare HTML for Conversion to a DOM Object” on page 13-108
- “Requirements for Converting HTML to a DOM Object” on page 13-110
- “Display Progress and Debugger Messages” on page 13-115
- “Create a Standalone Application from a Report Program” on page 13-118
- “Create a Microsoft Word Template” on page 13-119
- “Add Holes in a Microsoft Word Template” on page 13-120
- “Modify Styles in a Microsoft Word Template” on page 13-126
- “Create an HTML or PDF Template” on page 13-130
- “Add Holes in HTML and PDF Templates” on page 13-132
- “PDF and HTML Document Parts and Holes” on page 13-134
- “Modify Styles in HTML Templates” on page 13-137
- “Modify Styles in PDF Templates” on page 13-138
- “Create Chapters” on page 13-143
- “Create Page Layout Sections” on page 13-144
- “Create Page Footers and Headers” on page 13-148
- “Add Complex Page Numbers in Microsoft Word” on page 13-155
- “Functional Report” on page 13-157
- “Object-Oriented Report” on page 13-160
- “Report Formatting” on page 13-163
- “Create and Use a Custom Finder” on page 13-164
- “Resolve Error from Closing a PDF Document on a Cloud Drive” on page 13-188

## Create a Report Program

The MATLAB Report Generator includes classes that allow you to create report generator programs. These programs can generate Word, HTML, and PDF reports. The programs must include certain items and can include some optional items, both of which are listed here and described at each associated link. For information on Report API and how it compares to the Document Object Model (DOM), see “What Is a Reporter?” on page 1-3.

### Required Report Program Tasks and Elements

All report generator programs must:

- Create a report container. See “Create Report Container” on page 13-8.
- Create and add content to the container. See “Add Content to a Report” on page 13-10 and “Content Generation”. The content can be
  - Report API Reporters
  - DOM API objects
  - MATLAB objects (doubles, cell arrays, MATLAB tables, strings, and so on)
- Close the report container. See “Close a Report” on page 13-15.

### Optional Report Program Tasks and Elements

Optionally, in report generator programs, you:

- Import Report API classes, which enables using non-fully qualified Report API class names, for example, `TitlePage`, instead of `mlreportgen.report.TitlePage`. See “Import the API Packages” on page 13-7.
- Import DOM API classes, if the program adds DOM objects to the report, which enables using non-fully qualified DOM API class names.
- Configure reporters by setting their property values. See “Content Generation”.
- Add content to reporters by using the `add` method.

---

**Note** The only reporters you can both configure and add content to are the `Section` and `Chapter` reporters. The `Chapter` reporter is a subclass of the `Section` reporter.

---

- Display the report to see the generated report output. See “Display Report” on page 13-16.
- Display report progress messages to monitor report progress. See “Display Progress and Debugger Messages” on page 13-115.

### Report Generator Program Example

For example, this MATLAB code generates and displays a PDF report. It includes both required and optional items:

```
% Import report API classes (optional)
import mlreportgen.report.*

% Add report container (required)
```

```
rpt = Report('output','pdf');

% Add content to container (required)
% Types of content added here: title
% page and table of contents reporters
titlepg = TitlePage;
titlepg.Title = 'My Airplane';
titlepg.Author = 'Pilot A';
add(rpt,titlepg);
add(rpt,TableOfContents);

% Add content to report sections (optional)
% Text and formal image added to chapter
chap = Chapter('Plane Image');
add(chap,'Here is the plane:');
add(chap,FormalImage('Image',...
    which('b747.jpg'),'Height','5in',...
    'Width','5in','Caption','Boeing 747'));
add(rpt,chap);

% Close the report (required)
close(rpt);
% Display the report (optional)
rptview(rpt);
```



<b>Table of Contents</b>
<a href="#">Chapter 1. Plane Image</a> .....1



## Chapter 1. Plane Image

Here is the plane:



Figure 1.1. Boeing 747

## See Also

### More About

- “Create Report Container” on page 13-8
- “Add Content to a Report” on page 13-10
- “Construct a Report API or DOM API Object” on page 13-6
- “Close a Report” on page 13-15
- “Import the API Packages” on page 13-7
- “Content Generation”
- “Display Report” on page 13-16
- “Display Progress and Debugger Messages” on page 13-115
- “What Is a Reporter?” on page 1-3
- “Create a Standalone Application from a Report Program” on page 13-118
- “Templates”

## Construct a Report API or DOM API Object

The Report API and DOM API include a set of MATLAB functions, called constructors, for creating objects of various types, or classes.

The name of an object constructor is the name of the MATLAB class from which the API creates an object.

### Construct a Report API Object

The qualifier for Report API constructor names is `mlreportgen.report`. For example, the name of the constructor for a Report API chapter object is `mlreportgen.report.Chapter`. Some constructors do not require any arguments. Other constructors can take one or more arguments that typically specify its content and properties. For example, this code creates a chapter whose title is My Chapter.

```
chap = mlreportgen.report.Chapter('My Chapter');
```

A constructor returns a handle to the object it creates. Assigning the handle to a variable allows you to add content to the object or set its properties. For example, this code adds a section with a title to the chapter object `chap`.

```
append(chap,Section('Detailed Description'));
```

### Construct a DOM API Object

The qualifier for DOM API constructor names is `mlreportgen.dom`. For example, the name of the constructor for a DOM paragraph object is `mlreportgen.dom.Paragraph`. Some constructors do not require any arguments. Other constructors can take one or more arguments that typically specify its initial content and properties. For example, this code creates a paragraph whose initial content is Chapter 1.

```
para = mlreportgen.dom.Paragraph('Chapter 1.');
```

A constructor returns a handle to the object it creates. Assigning the handle to a variable allows you to append content to the object or set its properties. For example, this code appends content to the paragraph object `p`.

```
append(para,'In the Beginning');
```

### See Also

### Related Examples

- “Import the API Packages” on page 13-7

## Import the API Packages

MATLAB Report Generator function names, include either an `mlreportgen.report` prefix or an `mlreportgen.dom` prefix. You can omit the prefix if you insert these statements at the beginning of any program or function that uses the API.

```
import mlreportgen.report.*;  
import mlreportgen.dom.*;
```

Examples that refer to API objects and functions without the prefix assume that you have already imported the associated API package.

### See Also

### Related Examples

- “Create a Report Program” on page 13-3

## Create Report Container

### Create Report API Object to Hold Content

Every Report API program must create an `mreportgen.report.Report` object to hold report content. Use the `mreportgen.report.Report` constructor to create a `Report` object.

If you use the constructor without arguments, the Report API creates a PDF document named `untitled.pdf` in the current folder. To specify a name and location, use the path name of the report as the first argument of the constructor.

You can specify the type of report to generate by using the `type` argument. You can specify the type as `'html'`, `'docx'` (for Microsoft Word), `'pdf'` for PDF output, or `'html-file'` for single-file HTML output.

This `Report` constructor creates a document object called `myReport` for Word output.

```
d = Report('myReport', 'docx');
```

Using the `templatePath` argument, you can specify the path name of a custom template to use as a basis for formatting the report. Specify a template path if you want to base your report on a custom template that defines the appearance and structure of your report. The template type must match the document type. For example, this report constructor creates a report object for Word output using the template `myWordTemplate.dotx`.

```
d = Report('myReport', 'docx', 'myWordTemplate');
```

### Create a DOM Document Object to Hold Content

If you intend to use the DOM API alone (i.e., without using Report API objects) to generate a report, you must create an `mreportgen.dom.Document` object to hold the report content. Use the `mreportgen.dom.Document` constructor to create a `Document` object.

If you use the constructor without arguments, the DOM API creates an HTML document named `Untitled.htmx` in the current folder. To specify a name and location, use the path name of the report as the first argument of the constructor.

You can specify the type of report to generate by using the `type` argument. You can specify the type as `'html'`, `'docx'` (for Microsoft Word), `'pdf'` for PDF output, or `'html-file'` for single-file HTML output.

This `Document` constructor creates a document object called `myReport` for Word output.

```
d = Document('myReport', 'docx');
```

Using the `templatePath` argument, you can specify the path name of the template to use as a basis for formatting the report. Specify a template path if you want to base your report on a custom template that defines the appearance and structure of your report. The template type must match the document type. For example, this `Document` constructor creates a document object for Word output using the template `myWordTemplate.dotx`.

```
d = Document('myReport', 'docx', 'myWordTemplate');
```

## See Also

### Classes

`mlreportgen.report.Report` | `mlreportgen.dom.Document`

### Functions

`rptview`

## Related Examples

- “Create a Report Generator” on page 2-2
- “Construct a Report API or DOM API Object” on page 13-6
- “Create a Report Program” on page 13-3

## More About

- “Form-Based Reporting” on page 13-24
- “Templates for DOM API Report Programs” on page 13-22
- “What Is a Reporter?” on page 1-3

## Add Content to a Report

Report API and DOM API objects that serve as containers for report content have an `append` method that you use to add content to the objects. For example, objects of these classes have an `append` method:

- `mlreportgen.report.Report`
- `mlreportgen.report.Chapter`
- `mlreportgen.report.Section`
- `mlreportgen.dom.Document`
- `mlreportgen.dom.Paragraph`
- `mlreportgen.dom.Table`

The `append` methods take two arguments. The first argument is the object to append the content to and the second argument is the content to append to the object.

Here is an example that adds a paragraph to a chapter and a chapter to a report:

```
import mlreportgen.report.*
import mlreportgen.dom.*
rpt = Report("My Report");
ch = Chapter("My Chapter");
p = Paragraph("Hello World");
append(ch,p);
append(rpt,ch);
close(rpt);
rptview(rpt);
```

For some DOM API objects, such as paragraphs, you can specify the content when you create the object. Then, you can append more content to the object. For example, this code specifies the initial content for a paragraph, appends more content to the paragraph, and appends the paragraph to the document:

```
import mlreportgen.dom.*
d = Document("My Document");
p = Paragraph("Hello World.");
append(p," It's me");
append(d,p);
close(d);
rptview(d);
```

You can append DOM API objects and many built-in MATLAB objects, such as strings, character arrays, and cell arrays, to DOM API and Report API container objects. You can also append other Report API objects to Report API container objects. If the content that you try to append to an object is not supported for the object, the `append` method returns an error. For information about the content that you can append to a particular Report API or DOM API object, see the reference pages for the object and the `append` method of the object. For information about creating content, see “Content Generation”.

## See Also

### Functions

`mlreportgen.dom.Document` | `mlreportgen.dom.Paragraph` |  
`mlreportgen.report.Section` | `mlreportgen.report.Chapter` |  
`mlreportgen.report.Report`

### Related Examples

- “Create a Report Generator” on page 2-2
- “Construct a Report API or DOM API Object” on page 13-6
- “Add Content as a Group” on page 13-12

## Add Content as a Group

You can use a group to include the same content in different parts of a report. The DOM API clones the members of a group before appending them to another object.

This example shows the key code to include. After describing the steps involved in using a group, this example includes code for a complete report that uses a group.

- 1 Define the DOM objects that you want to include repeatedly in a report.

```
disclaimerHead = Heading(2, 'Results May Vary');
disclaimerIntro = Paragraph('The following results assume:');
disclaimerList = UnorderedList(...
    {'Temperature between 30 and 70 degrees F',...
     'Wind less than 20 MPH', 'Dry road conditions'});
```

- 2 Define a Group object that includes the DOM objects for the group. For example:

```
disclaimer = Group();
append(disclaimer, disclaimerHead);
append(disclaimer, disclaimerIntro);
append(disclaimer, disclaimerList);
```

- 3 Append the Group object in the place in the report where you want to repeat the content. For example, if the document object is doc:

```
append(doc, disclaimer);
```

This code builds a report based on this approach.

```
import mlreportgen.dom.*;
doc = Document('groupReport', 'html');
disclaimerHead = Heading(2, 'Results May Vary');
disclaimerIntro = Paragraph('The following results assume:');
disclaimerList = UnorderedList(...
    {'Temperature between 30 and 70 degrees F',...
     'Wind less than 20 MPH', 'Dry road conditions'});
disclaimer = Group();
append(disclaimer, disclaimerHead);
append(disclaimer, disclaimerIntro);
append(disclaimer, disclaimerList);
append(doc, disclaimer);
p1 = Paragraph('First set of results...');
p1.Bold = true;
p2 = Paragraph('more report content...');
p2.Bold = true;
append(doc, p1);
append(doc, p2);
append(doc, disclaimer);
close(doc);
rptview('groupReport', 'html');
```

## See Also

### Functions

clone



### **Classes**

`m\reportgen.dom.Group`

### **Related Examples**

- “Add Content to a Report” on page 13-10

## Output Types and Report Generator Packages

A Microsoft Word document packages all its contents, text, images, style sheets, and so on, in a single compressed `.docx` file.

For HTML documents, the DOM API defines an analogous packaging scheme, with an `htm` compressed file extension. By default, the DOM API generates HTML reports as `.htm` files.

To generate an HTML report in unzipped format, or in zipped and unzipped format, set the `PackageType` property of the `Document` object for a report to `'unzipped'` or `'both'`, respectively.

You can also output HTML as a single `.html` file.

PDF outputs a single `.pdf` file.

### See Also

#### Functions

`zipTemplate` | `unzipTemplate`

#### Classes

`mreportgen.dom.Document`

## Close a Report

The last step in creating a report with the DOM API is to close the report. A report must have content to produce an output file. Closing a report writes out any content that remains in memory and closes the report file. Use the `close` function.

```
d = Document('MyDoc');  
append(d, 'Hello World');  
close(d);
```

### See Also

`close`

### Related Examples

- “Create a Report Program” on page 13-3

## Display Report

The Report API and DOM API `rptview` functions allow you to display a generated report in an appropriate viewer:

- The Microsoft Word software for Word documents
- An HTML browser for HTML reports
- A PDF viewer for PDF reports

If an HTML report is in zipped format, `rptview` unzips a copy of the report in your temporary folder and displays the report's main HTML document in your default system browser.

To simplify your code, use the document output path as the argument to `rptview`. This example shows how to write your report program so you change only the value of the `doctype` variable to change the output type.

```
import mlreportgen.dom.*;
doctype = 'pdf';
d = Document('mydoc',doctype);

p = Paragraph('Hello World');
append(d,p);

close(d);
rptview(d.OutputPath);
```

Alternatively, you can specify the `rptview` function with two arguments:

- The path of the report — If you specify the file extension, you do not need to specify the second argument for output type.
- The output type — 'html', 'pdf', or 'docx'.

Use 'pdf' with a report formatted for Word to convert the Word document to PDF and open it in a PDF viewer.

## See Also

### Functions

`rptview`

## Related Examples

- “Create a Report Program” on page 13-3

# Report Formatting Approaches

You can format your report using style sheets, format objects, format properties, or a combination of these approaches.

## Style Sheets in Templates

The DOM API comes with default templates for each output type for formatting your report as it generates. You can customize these templates to specify the default formatting and layout of your reports. See “Templates for DOM API Report Programs” on page 13-22.

Use style sheets in a template to describe the default formatting of document objects like paragraphs, headers, and tables. A style sheet is a collection of formatting styles. A style is a named collection of formats for a particular type of object or, in the case of HTML and PDF, for a particular type of object that appears in a particular context in your document. For example, you can define a paragraph style `MyPara` that uses one set of formats, such as font size, emphasis, and font family. You define another paragraph style named `YourPara` that uses a different set of formats. When you write your report program, you assign the style to a paragraph object by name. For an example, see “Use Style Sheet Styles” on page 13-19.

## Format Objects

A format object is a MATLAB object that defines the properties and functions of a document format, such as a font family or size. The DOM API provides a set of constructors for creating format objects corresponding to most of the formatting options available in HTML, Word, and PDF documents. Most DOM document objects include a `Style` property that you can set to a cell array of format objects. You can use format objects with the document object `Style` property to format the object. For example, this code uses format objects to specify the style of a warning paragraph.

```
import mlreportgen.dom.*

p = Paragraph('Danger!');
p.Style = {Color('red'),FontFamily('Arial'),FontSize('18pt')};
```

It is a best practice to set the `Style` property by concatenating the existing value of the `Style` property and the cell array of format objects that you are adding. For example:

```
import mlreportgen.dom.*

p = Paragraph('Danger!');
p.Style = [p.Style {Color('red'),FontFamily('Arial'),FontSize('18pt')}];
```

This practice prevents inadvertent removal of format objects that you previously added or that the DOM API added to synchronize the `Style` property with the format properties. See “Format Properties” on page 13-18.

You can assign the same array of format objects to more than one DOM document object. This technique allows you to create a programmatic equivalent of a template style sheet. For example:

```
import mlreportgen.dom.*

warning = {Color('red'),FontFamily('Arial'),FontSize('18pt')};
p = Paragraph('Danger!');
p.Style = [p.Style warning];
```

```
p = Paragraph('Caution!');  
p.Style = warning;
```

The DOM API allows you to assign any format object to any document object, regardless of whether the format applies. If the format does not apply, it is ignored.

## Format Properties

Most DOM objects have a set of properties corresponding to the format options most commonly used for an object of that class. For example, this code sets the font and color of text in a paragraph, using the Color, FontFamily, and FontSize format properties of a Paragraph object.

```
import mlreportgen.dom.*  
  
p = Paragraph('Danger!');  
p.Color = 'red';  
p.FontFamilyName = 'Arial';  
p.FontSize = '18pt';
```

## See Also

### More About

- “Use Style Sheet Styles” on page 13-19
- “Format Inheritance” on page 13-21

## Use Style Sheet Styles

A style is a collection of formats that define the appearance of a document object, such as a paragraph, table, or list. You can define and name styles in templates and then assign the names to paragraphs, tables, and other document elements in your report program. The style determines how the document object renders in the output.

In a Word template, you can define styles and save styles that belong together in a style sheet (also called a style set). In an HTML template, you define styles in a cascading style sheet (CSS) file. You define styles for PDF documents in a CSS file, using a subset of CSS. See “Modify Styles in PDF Templates” on page 13-138.

You can apply style sheet styles to document objects using the `StyleName` property in your report program using this workflow.

- 1 In the template you are using with the report, define or modify styles.
- 2 In a DOM report, create a `Document` object that uses the template that contains your styles.
- 3 For the objects that you want to format with your styles, set the `StyleName` property to match the name of the style in the template.

For example, this code assigns a style named `Warning` to a paragraph object. It assumes that you have defined the `Warning` style in a Word template named `MyTemplate.dotx`. Assigning the `Warning` style to the DOM paragraph object applies the `Warning` style in the template to the paragraph when you generate the report.

```
d = Document('MyDoc', 'docx', 'MyTemplate');
p = Paragraph('Use care when unplugging this device.');
```

`p.StyleName = 'Warning';`  
`append(d,p);`  
`close(d);`

---

**Tip** Some document object constructors allow you to specify the value of the `StyleName` property as an argument. For example, this paragraph applies the style `Warning` to the paragraph containing the specified text.

```
p = Paragraph('Use care when unplugging this device', 'Warning');
```

---

## See Also

### Related Examples

- “Create a Microsoft Word Template” on page 13-119
- “Modify Styles in a Microsoft Word Template” on page 13-126
- “Create an HTML or PDF Template” on page 13-130
- “Modify Styles in HTML Templates” on page 13-137
- “Modify Styles in PDF Templates” on page 13-138

**More About**

- “Report Formatting Approaches” on page 13-17
- “Format Inheritance” on page 13-21



## Format Inheritance

The DOM API allows you to use template-based styles and format object-based styles (or equivalent format properties) to specify the appearance of an object. If you set the `StyleName` and the `Style` property of an object, the formats in the `Style` property override corresponding formats specified by the template-based style of the `StyleName` property. Consider, for example, this code.

```
import mlreportgen.dom.*;
d = Document('MyDoc', 'docx', 'MyTemplate');
p = Paragraph('Danger!');
p.StyleName = 'Warning';
p.Style = {Color('red')};
append(d,p);
close(d);
```

Suppose that the `Warning` style defines the color of a warning as yellow. In that case, the setting of the `Style` property on the paragraph overrides the color specified by the `StyleName` setting.

If a document object does not specify a value for `StyleName`, it inherits any formats that it does not specify from its container. The container inherits any formats that it does not specify from its container, and so on, all the way to the top of a container hierarchy. Format inheritance allows you to use a single statement to assign a format for all the objects contained by a container. For example, this code uses a single `Style` property to assign a color to all the entries in a table.

```
import mlreportgen.dom.*;
d = Document('MyDoc');
tableArray = {'a', 'b'; 'c', 'd'};
table = append(d, tableArray);
table.Style = {Color('blue')};
close(d);
rptview(d.OutputPath);
```

### See Also

#### More About

- “Use Style Sheet Styles” on page 13-19
- “Report Formatting Approaches” on page 13-17

## Templates for DOM API Report Programs

The DOM API comes with default templates for each output type for formatting your report as it generates. Templates are useful for providing default design formats so that you do not need to specify them in your report. This approach is helpful if several reports have the same look, which is typical in most organizations. In your report program, you refer by name to the template and its styles and layouts. When your report generates, the template determines the appearance of the document objects.

Templates also enable form-based document generation. You can define fixed content and holes (blanks) in your template. Your report program can fill the holes with content, such as text or images. See “Form-Based Reporting” on page 13-24.

Another advantage of using templates is for maintenance. If your report design changes, you change only the template and not all the programs that use that design.

Using templates also keeps your report program smaller, because you do not need to specify properties for each object you create. For reports that are hundreds of pages, using templates might also improve performance.

You can create a copy of the default templates and customize them to specify the default formatting and layout of your reports. For the template to take effect, your report program must refer to your template and specify the style names and document parts to use.

You can create a copy of the default templates using the `mlreportgen.dom.Document.createTemplate` method. The default templates can serve as a starting point for your template.

### Template Packages

All DOM templates, except for single-file HTML templates, consist of document, style sheet, and image files zipped into packages based on the Open Packaging Convention (OPC). A single-file HTML template embeds style sheets and images as HTML elements in the HTML document. You can use Microsoft Word to edit Word templates (identified by a `.dotx` extension) directly. You can also edit single-file HTML templates directly using any text or HTML editor.

To edit multifile HTML templates (identified by an `.htmtdx` extension) and PDF templates (identified by a `.pdftx` extension), you must first unzip them. You can optionally rezip an edited HTML or PDF template before using it to generate a report. The DOM API provides functions for zipping and unzipping multifile HTML and PDF templates: `zipTemplate` and `unzipTemplate`.

### Styles

You can use styles defined in templates to format paragraphs, text, tables, lists, and so on. You can modify styles or create your own. See “Use Style Sheet Styles” on page 13-19.

Word templates include standard Word styles, such as Normal, Heading 1, and Title. You create and modify styles using standard Word techniques. See “Modify Styles in a Microsoft Word Template” on page 13-126.

HTML and PDF templates define styles using CSS properties in template files that end with `.css`. For details, see “Modify Styles in HTML Templates” on page 13-137 and “Modify Styles in PDF Templates” on page 13-138.

## Page Layout

You can use templates to define the page layout of Word and PDF reports, including the size, orientation (portrait or landscape), margins, and page headers and footers. You can use a template to define different page layouts for different sections of a document. See “Create Page Layout Sections” on page 13-144.

You can also define page layouts programmatically or use a combination of layouts that are defined programmatically and in a template.

## Document Part Templates

A document part template is a template for a repeatable structure in your report. You can insert an instance of a document part in your report from your report program using a `DocumentPart` object. You create document part templates in a document part template library.

For Word templates, you define document part templates and store them in the Word Quick Parts Gallery, which serves as the library. The default template does not include any document part templates. To create them, see “Create a Microsoft Word Document Part Template Library” on page 13-27.

For HTML and PDF, the default template contains a document part template library file named `docpart_templates.html`. This file creates the library and contains some default document part templates. You can modify or delete the supplied document part templates and add your own. See “Create an HTML Document Part Template Library” on page 13-30 and “Create a PDF Document Part Template Library” on page 13-32.

## See Also

`unzipTemplate` | `zipTemplate` | `mlreportgen.dom.Document.createTemplate`

## Related Examples

- “Use Style Sheet Styles” on page 13-19
- “Modify Styles in PDF Templates” on page 13-138
- “Modify Styles in HTML Templates” on page 13-137
- “Create a Microsoft Word Document Part Template Library” on page 13-27
- “Create an HTML Document Part Template Library” on page 13-30
- “Create a PDF Document Part Template Library” on page 13-32

## Form-Based Reporting

The DOM API supports a form-based approach to report generation. You can create a template that defines the fixed content of the form, interspersed with holes (blanks). Your DOM report program fills these holes with generated content.

### See Also

#### Related Examples

- “Fill the Blanks in a Report Form” on page 13-25
- “Use Subforms in a Report” on page 13-26
- “Create a Microsoft Word Template” on page 13-119
- “Add Holes in a Microsoft Word Template” on page 13-120
- “Create an HTML or PDF Template” on page 13-130
- “Add Holes in HTML and PDF Templates” on page 13-132

## Fill the Blanks in a Report Form

When you create a form template, you associate an ID with each hole in the template. The ID allows you to navigate the holes in a form, using the DOM `moveToNextHole` function.

The first time you execute the `moveToNextHole` function, the DOM API copies to the output document all of the text in the template up to the first hole. At this point, you can start adding content to the output document using the DOM `append` function, thereby filling in the first hole.

The next time you execute the `moveToNextHole` function, the DOM API copies all the text between the first and second hole in the template to the output document. You can then fill in the second hole by appending content to the output document. In this way, you generate the output document by copying the content from the template and filling in all its holes.

For example, this function generates a report from a Word template that has holes named `Title`, `Author`, and `Content`. The arguments `title`, `author`, and `content`, are assumed to be character vectors.

```
function makerpt(title,author,content,rptname,rpttemplate)
    import mlreportgen.dom.*
    rpt = Document(rptname,'docx',rpttemplate);

    while ~strcmp(rpt.CurrentHoleId,'#end#')
        switch rpt.CurrentHoleId
            case 'Title'
                append(rpt,title);
            case 'Author'
                append(rpt,author);
            case 'Content'
                append(rpt,content);
        end
        moveToNextHole(rpt);
    end

    close(rpt);
```

### See Also

#### Functions

`moveToNextHole`

### Related Examples

- “Use Subforms in a Report” on page 13-26
- “Create a Microsoft Word Template” on page 13-119
- “Add Holes in a Microsoft Word Template” on page 13-120
- “Create an HTML or PDF Template” on page 13-130
- “Add Holes in HTML and PDF Templates” on page 13-132

### More About

- “Form-Based Reporting” on page 13-24

## Use Subforms in a Report

A document part is a form that you can add to a document or to another document part. Document parts simplify generating sections of a report that have the same form, such as sections that report on the results of a series of tests or the performance of a series of financial portfolios. Use a similar approach as you do for main document forms.

- 1** Create a template that defines the form of the document part.
- 2** For each section:
  - a** Create an `m1reportgen.dom.DocumentPart` object.
  - b** Fill in the holes.
  - c** Append the part to the main document.

For an example of a report that uses subforms, open the “Functional Report” on page 13-157 example.

---

**Tip** The DOM API allows you to store the templates for document parts in the main template for a report. This capability allows you to use a single template file to supply all the templates required for a report. For details, see “Create a Microsoft Word Document Part Template Library” on page 13-27.

---

### See Also

#### Functions

`moveToNextHole`

#### Classes

`m1reportgen.dom.DocumentPart`

### Related Examples

- “Fill the Blanks in a Report Form” on page 13-25
- “Create a Microsoft Word Template” on page 13-119
- “Add Holes in a Microsoft Word Template” on page 13-120
- “Create an HTML or PDF Template” on page 13-130
- “Add Holes in HTML and PDF Templates” on page 13-132

### More About

- “Form-Based Reporting” on page 13-24

## Create a Microsoft Word Document Part Template Library

### In this section...

“Create Document Part Template Library in a Word Template” on page 13-27

“Word Document Part List Limitations” on page 13-28

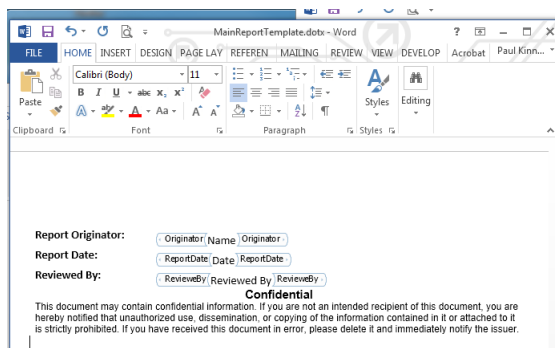
A document part template library is a set of document part templates stored by name in a template file. Document part template libraries allow you to store all the templates for a report in a single template file, for example, the main template file of a report. You can create additional template files, each with a specific purpose, and create document part template libraries in them.

Using the DOM API, you can create an instance of a document part based on a template stored in a library by specifying the name of the template in the document part constructor. For form-based reports that you create using the Report Explorer, create an instance by specifying the template file and the name of the subform template from the document part template library.

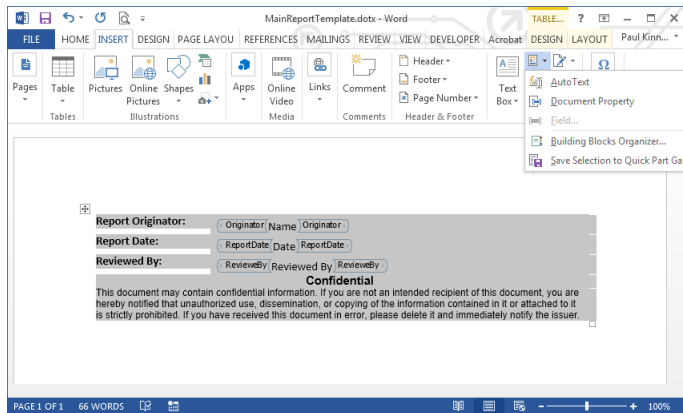
### Create Document Part Template Library in a Word Template

You can use the Quick Parts Gallery in Word to create a document part template library in the main template of a report. A Quick Part Gallery is a collection of reusable pieces of preformatted content, called quick parts, that are stored in the document. You can use quick parts as templates for DOM DocumentPart objects.

- 1 Open the Word template in which you want to create the document part template.
- 2 In the template, create the Word content to serve as a prototype for the document part template. (You delete the prototype after copying it to the Quick Part Gallery.) The document part template content that you create can contain holes and page layout sections, and other types of Word content. For example:



- 3 Select the content that you have created for the document part template.
- 4 On the **Insert** tab, click the **Explore Quick Parts** button. Select **Save Selection to the Quick Parts Gallery**.



- 5 In the Create New Building Block dialog box, in the **Name** field, enter a unique name for the template. Use this name in the constructor of a DocumentPart object.
- 6 For the first document part template you create in the template file, in the **Category** list, click Create New Category. Create a category named mlreportgen. Then select mlreportgen from the **Category** list.

Otherwise, select mlreportgen from the **Category** list.

- 7 In the **Description** field, enter a template description and click **OK**.
- 8 Delete the content that served as the prototype for the document part template.
- 9 Save the template file.

### Modify Document Part Template in Quick Part Gallery

You can modify a document part template stored in the Quick Part Gallery.

- 1 Open the Word template that contains the document part template.
- 2 Click in the template where you want to create an instance of the document part template.
- 3 On the **Insert** tab, click the **Explore Quick Parts** button.
- 4 In the Quick Part Gallery, to create an instance, select the document part template you want to modify.
- 5 Edit the instance.
- 6 Select the modified instance. On the **Insert** tab, click **Explore Quick Parts** and select **Save Selection to the Quick Part Gallery**.
- 7 In the Create New Building Block dialog box, enter the name of the document part template you modified and select the mlreportgen category. Respond to the prompt to overwrite the previous version.
- 8 Delete the instance in the template document, and save and close the template.

### Word Document Part List Limitations

The DOM API does not support use of lists in the fixed content of a document part template. Such lists can appear with incorrect formatting in the output document. To include a list in a document part, generate the list programmatically, that is, append an `OrderedList` or `UnorderedList` object to the part where you want the list to appear. If you want to apply a style to the list, you must use a list style defined in your main document.



## See Also

### Classes

`mreportgen.dom.DocumentPart`

## Related Examples

- “Fill the Blanks in a Report Form” on page 13-25
- “Create a Microsoft Word Template” on page 13-119
- “Add Holes in a Microsoft Word Template” on page 13-120
- “Add Holes in HTML and PDF Templates” on page 13-132
- “Create a PDF Document Part Template Library” on page 13-32
- “Create an HTML Document Part Template Library” on page 13-30

## More About

- “Form-Based Reporting” on page 13-24

## Create an HTML Document Part Template Library

In the default template package, the file `docpart_templates.html` defines the library and some default document part templates. In your document part template library, create the document parts that you want to reuse throughout your report. You can create a part template for any part of your document that you want to repeat without redefining it programmatically.

A document part template typically consists of fixed content and holes. You can use standard HTML elements to define your templates. You can also use the `<toc>` element for a table of contents. For details, see “Create a Table of Contents” on page 13-92.

Use this workflow to work on your document part template library:

- 1 Unzip the template package containing the part template library file.
- 2 Open the document part templates file, named `docpart_templates.html` by default, in an HTML or text editor.
- 3 Edit the file as needed using the elements described in “HTML Document Part Template Library Structure” on page 13-30.
- 4 Add any styles that support the document part templates in a `.css` file in the template package. See “Modify Styles in HTML Templates” on page 13-137.
- 5 Save the library files you edited.
- 6 Repackage the template using `ziptemplate`.

### HTML Document Part Template Library Structure

You create your document part library using the `<dplibrary>` element. Add a `<dplibrary>` element inside a `<body>` element. Your template package can have only one `<dplibrary>` element.

Use `<dptemplate>` elements inside a `<dplibrary>` element for each document part template that you want to create. You can create as many document part templates as you need.

This code shows the basic structure of a document part library. The `<dptemplate>` element has the attribute `name`, which you set to the name to use when you call the document part from your report program. The name is equivalent to the name of the part in the Quick Parts Gallery in Word. If you are creating templates for multiple outputs, use the same name in both places.

```
<body>
  <dplibrary>
    <dptemplate name="myFirstDocPartTemp">
      [Document part template content--holes and fixed content]
    </dptemplate>
  </dplibrary>
</body>
```

### See Also

`zipTemplate` | `unzipTemplate`

## **Related Examples**

- “Create an HTML or PDF Template” on page 13-130
- “Add Holes in HTML and PDF Templates” on page 13-132
- “Create a PDF Document Part Template Library” on page 13-32
- “Create a Microsoft Word Document Part Template Library” on page 13-27

## Create a PDF Document Part Template Library

In the default template package, the file `docpart_templates.html` defines the library and some default document part templates. In your document part template library, create the document parts that you want to reuse throughout your report. You can create a part template for any part of your document that you want to repeat without redefining it programmatically.

A document part template typically consists of fixed content and holes. It can also include page layout elements that describe the page size, margins, and orientation as well as page headers and footers. You create PDF document part template libraries using DOM API HTML elements provided for this purpose and a subset of HTML elements.

Use this workflow to work on your document part template library.

- 1 Unzip the template package containing the part template library file.
- 2 Open the document part templates file, named `docpart_templates.html` by default, in an HTML or text editor.
- 3 Edit the file as needed using the elements described in “PDF Document Part Template Library Structure” on page 13-32.
- 4 Add any styles that support the document part templates in a `.css` file in the template package. See “Modify Styles in PDF Templates” on page 13-138.
- 5 Save the library files you edited.
- 6 Repackage the template using `ziptemplate`.

### PDF Document Part Template Library Structure

You create your document part library using the `<dplibrary>` element. Add a `<dplibrary>` element inside a `<body>` element in your `docpart_template.html` file. Your template package can have only one `<dplibrary>` element.

Use `<dptemplate>` elements inside a `<dplibrary>` element for each document part template that you want to create. You can create as many document part templates as you need.

This code shows the basic structure of a document part library. The `<dptemplate>` element has the attribute `name`, which you set to the name that you use to call the document part. The name is equivalent to the name of the part in the Quick Parts Gallery in Word. If you are creating templates for multiple outputs, use the same name in both places.

```
<body>
  <dplibrary>
    <dptemplate name="myFirstDocPartTemp">
      [Document part template content here--
      holes, fixed content, page layout information, and HTML]
    </dptemplate>
  </dplibrary>
</body>
```

## Document Part Template Library Contents

You can use DOM API HTML elements and a subset of standard HTML elements to create PDF document part templates. For examples that show how to use the DOM API HTML elements, see:

- “Create a Table of Contents” on page 13-92
- “Automatically Number Document Content” on page 13-101
- “Add Holes in HTML and PDF Templates” on page 13-132
- “Create Page Layout Sections” on page 13-144
- “Create Page Footers and Headers” on page 13-148
- “PDF and HTML Document Parts and Holes” on page 13-134
- “Create a Page Reference” on page 13-83

### DOM API HTML Elements

In addition to the `<dplibrary>` and `<dptemplate>` elements that you use to define the library and the document parts, you can use these DOM API HTML elements in your PDF templates.

Purpose	Element	Attributes	Values
Page layout	layout	style	page-margin: top left bottom right header footer gutter; page-size: height width orientation; page-border: width style color margin surround-header surround-footer
		first-page-number	Number of first page in the layout
		page-number-format	n or N for numeric, a, A, i, I
		section-break	Where to start section for this layout: Odd Page, Even Page, or Next Page
Page header	pheader	type	default, first, even
		template-name	Document part template that defines the header
Page footer	pfooter	type	default, first, even
		template-name	Document part template that defines the footer
Page number format (same as first-page-number and page-number-format on layout)	pnumber	format	n or N for numeric, a, A, i, I
		initial-value	The number for the first page in the layout that uses this element
Hole	hole	id	ID that identifies hole by name
		default-style-name	Style sheet style to use when style is not set programmatically

Purpose	Element	Attributes	Values
Table of contents	toc	number-of-levels	Number of heading levels to include in TOC
		leader-pattern	Leader pattern to use: dots, space, period, or space
Automatic numbering	autonumber	stream-name	Name of the stream specified by a counter-increment style
Current page number	page	No attributes	
Total number of pages in document	numpages	No attributes	
Page break	pagebreak	No attributes	
Numeric reference to page where target is located	pageref	target	ID of target; create target in your report using <code>mlreportgen.dom.LinkTarget</code>
Insert content of a heading or other style into a page header or footer (for running headers and footers)	styleref	No attributes	Inserts content of nearest h1 element
		style-name	Name of the style with content to insert in the header or footer
		outline-level	Outline level of style with content to insert in the header or footer
Insert a watermark image in a page layout	watermark	src	Path of the source file to use as the watermark image. Store the watermark image in the template package. See "Watermarks in PDF Page Layouts" on page 13-145.
		width	Width to scale watermark image, in the form <code>valueUnits</code> . Possible values for units are <code>px</code> , <code>in</code> , <code>cm</code> , <code>mm</code> , <code>pc</code> , and <code>pt</code> .
		height	Height to scale watermark image, in the form <code>valueUnits</code>

For detailed information on the attributes, see the properties for these corresponding DOM API classes.

- `mlreportgen.dom.PDFPageLayout`
- `mlreportgen.dom.PDFPageHeader`
- `mlreportgen.dom.PDFPageFooter`
- `mlreportgen.dom.TOC`
- `mlreportgen.dom.AutoNumber`
- `mlreportgen.dom.PageRef`
- `mlreportgen.dom.StyleRef`

- mlreportgen.dom.Watermark

### Standard HTML Elements

You can use these standard HTML elements in PDF templates.

HTML Element	Attributes
a	class, style, href, name
address	class, style
b	class, style
big	class, style
blockquote	class, style
body	class, style
br	n/a
center	class, style
cite	class, style
code	class, style
dd	class, style
del	class, style
dfn	class, style
div	class, style
dl	class, style
dt	class, style
em	class, style
font	class, style, color, face, size
h1, h2, h3, h4, h5, h6	class, style, align
hr	class, style, align
i	class, style
ins	class, style
img	class, style, src, height, width
kbd	class, style
li	class, style
mark	class, style
nobr	class, style
ol	class, style
p	class, style, align
pre	class, style
s	class, style
samp	class, style

HTML Element	Attributes
small	class, style
span	class, style
strike	class, style
strong	class, style
sub	class, style
sup	class, style
table	class, style, align, bgcolor, border, cellpadding, cellspacing, frame, rules, width
tbody	class, style, align, valign
tfoot	class, style, align, valign
thead	class, style, align, valign
td	class, style, bgcolor, height, width, colspan, rowspan, align, valign, nowrap
th	class, style, bgcolor, height, width, colspan, rowspan, align, valign, nowrap
tr	class, style, align, bgcolor, valign
tt	class, style
u	class, style
ul	class, style
var	class, style

For information about these elements, see <https://developer.mozilla.org/en-US/docs/Web/HTML/Element>.

## See Also

zipTemplate | unzipTemplate

## Related Examples

- “Create an HTML or PDF Template” on page 13-130
- “Add Holes in HTML and PDF Templates” on page 13-132
- “Create an HTML Document Part Template Library” on page 13-30
- “Create a Microsoft Word Document Part Template Library” on page 13-27



## Simplify Filling in Forms

The object-oriented approach allows you to use the DOM `fill` method to simplify form-based reporting. The `fill` method is intended for instances of classes derived from the `mlreportgen.dom.Document` or `mlreportgen.dom.DocumentPart` class. It assumes that for each hole in a document or document part template, the derived class defines a method having this signature:

```
fillHoleID(obj)
```

The `HoleID` part of the signature is the ID of a hole defined by the document or document part template. The `obj` argument is an instance of the derived class. For example, supposed that a template defines a hole named `Author`. Then the derived class defines a method name `fillAuthor` to fill the `Author` hole. Assuming that the derived class defines methods for filling the holes, the `fill` method moves from the first hole in the document or part to the last, invoking the corresponding `fillHoleID` method to fill each hole.

The `fill` method eliminates the need for a report program to loop explicitly through the holes in a document or document part's template. The report need only invoke the document or part `fill` method. For example, suppose that you have derived a report class, name `MyReport`, from the `mlreportgen.dom.Document` class and that this derived class defines methods for each of the holes defined by the report template, based on data supplied in its constructor. Then, you need only three lines to generate an instance of `MyReport`:

```
function makeReport(rptdata)
rpt = MyReport(rptdata);
fill(rpt);
close(rpt);
```

For an example of a forms-based, object-oriented report program, in the **Examples** pane of the MATLAB Report Generator documentation, open the "Object-Oriented Report" on page 13-160 example .

### See Also

#### Functions

`moveToNextHole`

#### Classes

`mlreportgen.dom.DocumentPart`

### Related Examples

- "Use Subforms in a Report" on page 13-26
- "Fill the Blanks in a Report Form" on page 13-25
- "Create a Microsoft Word Template" on page 13-119
- "Add Holes in a Microsoft Word Template" on page 13-120
- "Create an HTML or PDF Template" on page 13-130
- "Add Holes in HTML and PDF Templates" on page 13-132

**More About**

- “Form-Based Reporting” on page 13-24

## Create and Format Text

### In this section...

“Create Text” on page 13-39

“Create Special Characters” on page 13-39

“Append HTML or XML Markup” on page 13-39

“Format Text” on page 13-40

### Create Text

You can create text by appending a character vector to a document, paragraph, table entry, or list item. The DOM `append` function converts the character vector to a `Text` object, appends it, and returns the `Text` object. Use the `Text` object to format the text. You can also create a text object directly and append it to a document. This example:

- Creates the `Text` object `t1` by appending 'Hello' to the document
- Uses a `Text` constructor to create a `Text` object and append the text 'World' to the document

```
import mlreportgen.dom.*
d = Document('mydoc', 'html');

t1 = append(d, 'Hello');

append(d, Text('World'));

close(d);
rptview(d.OutputPath);
```

### Create Special Characters

You can define special characters, such as the British pound symbol, to include in a report by creating an `mlreportgen.dom.CharEntity` object. Specify a name of a character entity listed at [https://en.wikipedia.org/wiki/List\\_of\\_XML\\_and\\_HTML\\_character\\_entity\\_references](https://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references). For example:

```
import mlreportgen.dom.*;
d = Document('test', 'html');

p = Paragraph(CharEntity('pound'));
append(d, p);
append(p, '3');

close(d);
rptview(d.OutputPath);
```

### Append HTML or XML Markup

To append HTML markup to an HTML document or Microsoft Word XML markup to a Word document, use an `mlreportgen.dom.RawText` object. This technique is useful for creating HTML or Word elements that the DOM API does not support directly. This example shows how to create a `RawText` object to append HTML markup.

```
import mlreportgen.dom.*;
d = Document('test', 'html');

append(d, RawText('<em>Emphasized Text</em>'));

close(d);
rptview('test', 'html');
```

## Format Text

You can format text programmatically, using either DOM format objects or `Text` object format properties. You can also use template styles. For information about these formatting techniques and format inheritance, see “Report Formatting Approaches” on page 13-17.

### Format Text Programmatically

You can use format objects to format `Text` objects or format properties to specify commonly used text formats. This example uses:

- A `FontFamily` format object to specify the primary and backup font
- The `Bold` format property to specify text weight

```
import mlreportgen.dom.*;
d = Document('test', 'html');

t = append(d, 'Bold Arial text');

fontFamily = FontFamily('Arial');
fontFamily.BackupFamilyNames = {'Helvetica'};
t.Style = {fontFamily};

t.Bold = true;

close(d);
rptview(d.OutputPath);
```

Use these format objects and format properties to format text.


Formatting	Format Object	Format Property
Font	FontFamily	FontFamilyName
Backup font (HTML only)	FontFamily	n/a
Complex script font (for example, Arabic)	FontFamily	n/a
East Asian font	FontFamily	n/a
Font size	FontSize	FontSize
Foreground color	Color	Color
Background color	BackgroundColor	BackgroundColor
Bold	Bold	Bold
Italic	Italic	Italic
Subscript or superscript	VerticalAlign	n/a

Formatting	Format Object	Format Property
Strike through	Strike	Strike
Underline type (single, double, etc.)	Underline	Underline
Underline color	Underline	n/a
Preserve white space	WhiteSpace	WhiteSpace
Display as specified	Display	n/a

### Format Text Using Microsoft Word Style Sheets

You can format a paragraph using a style defined in the Word template used to generate the report.

To define a text style in a Word template, start by using these steps:

- 1 Open the Word template used with the report.
- 2 Open the **Styles** pane.
- 3 Click the **Manage Styles** button .
- 4 Click **New Style**.
- 5 In the Create New Style from Formatting dialog box, set **Style type** to Character or Linked (paragraph and character).

For more information about working with Word styles, see “Modify Styles in a Microsoft Word Template” on page 13-126.

### Format Text for HTML and PDF Using Style Sheets

You can format text using a style defined in the template used to generate the report. Apply a template style to a Text object either as the second argument in a Text object constructor or by setting the StyleName property to a template style.

To define the style, use cascading style sheet (CSS) syntax. Use a selector on the span element to specify the style name. This CSS defines a style named Pass.

```
span.Pass {
  font-family: "Times New Roman", Times, serif;
  color: green;
}
```

You can use any CSS properties and selectors in HTML templates. For PDF templates, you can use a subset of CSS properties and selectors. See “Modify Styles in PDF Templates” on page 13-138.

### Apply a Style to a Text Object

Apply a template style to a Text object either as the second argument in a Text object constructor or by setting the StyleName property to a template style. Suppose you have defined styles named Body, Pass, and Fail in the template for your report. You can then apply the styles.

```
import mlreportgen.dom.*;
passed = rand(1) >= 0.5;
rpt = Document('MyReport', 'html', 'MyTemplate');
```

```
t1 = Text('Test status: ');
t1.StyleName = 'Body';
t1.WhiteSpace = 'preserve';

if passed
    status = 'Passed';
    statusStyle = 'Pass';
else
    status = 'Failed';
    statusStyle = 'Fail';
end

t2 = Text(status,statusStyle);
statusPara = Paragraph(t1);
append(statusPara,t2);
append(rpt, statusPara);

close(rpt);
rptview(rpt.OutputPath);
```

### Override Template Formats

You can use programmatic formats to override the formats defined in a template-based style. Suppose that you define a style named `AlertLevel` in your template that sets the color to green. You can override the style in your report program to set a color based on the current alert level:

```
t = Text('Danger!', 'AlertLevel');
t.Color = 'red';
```

### See Also

#### Classes

`mlreportgen.dom.Text` | `mlreportgen.dom.CharEntity` | `mlreportgen.dom.FontFamily` |  
`mlreportgen.dom.FontSize` | `mlreportgen.dom.Bold` | `mlreportgen.dom.Italic` |  
`mlreportgen.dom.Underline` | `mlreportgen.dom.Strike`

### Related Examples

- “Add Content to a Report” on page 13-10
- “Modify Styles in HTML Templates” on page 13-137

### More About

- “Report Formatting Approaches” on page 13-17

## Format Numbers

By default, the DOM API uses the maximum number of digits needed to accurately represent a number as text in a report. To control the number of digits used to represent a number, specify a number format using these approaches:

- Set the default number format by calling the `mlreportgen.dom.setDefaultNumberFormat` function.
- Override the default format for one number by representing the number as an `mlreportgen.dom.Number` object and including an `mlreportgen.dom.NumberFormat` object in the `Style` property of the `Number` object.
- Override the default format for all numbers in a document element, such as a paragraph, table, or list, by including an `mlreportgen.dom.NumberFormat` object in the `Style` property of the object that represents the element.

### Set the Default Number Format

To set the default format specification that the DOM API uses to format numeric data, use the `mlreportgen.dom.setDefaultNumberFormat` function. Provide a format specification that is valid for `sprintf` and that uses the `%f`, `%e`, `%E`, `%g`, or `%G` operator. For example, this code specifies four digits after the decimal point:

```
mlreportgen.dom.setDefaultNumberFormat("%0.4f");
```

The default format applies for the duration of the MATLAB session and applies to numbers in document elements, such as paragraphs, tables, and lists, and to numbers represented as `mlreportgen.dom.Number` objects. For example, this code uses the default format to represent `pi` in a report:

```
import mlreportgen.dom.*
setDefaultNumberFormat("%0.4f");
d = Document("myDoc", "pdf");
append(d, Paragraph(pi));
close(d);
rptview(d);
```

In the report, `pi` is generated as 3.1416.

### Specify the Format for One Number

To specify the format for one number:

- Represent the number as an `mlreportgen.dom.Number` object.
- Specify the number format in an `mlreportgen.dom.NumberFormat` object and include the `NumberFormat` object in the `Style` property of the `Number` object.

For example, this code uses the default number format for the first instance of `pi` and overrides the default format for the second instance of `pi`:

```
import mlreportgen.dom.*
setDefaultNumberFormat("%0.4f");
d = Document("myDoc1", "pdf");
```

```
p1 = Paragraph("pi with default format: ");
append(p1,pi);
append(d,p1);

p2 = Paragraph("pi with number format: ");
n = Number(pi);
n.Style = {NumberFormat("%0.2f")};
append(p2,n);
append(d,p2);

close(d);
rptview(d);
```

Here are the numbers in the report:

```
pi with default format: 3.1416
pi with number format: 3.14
```

## Specify the Number Format for a Paragraph, Table, or List

You can specify the number format for all the numbers in a document element, such as a paragraph, table, or list by including an `mlreportgen.dom.NumberFormat` object in the `Style` property of the object that represents the document element. For example, this code specifies that the numbers in the first paragraph use the default format and that the numbers in the second paragraph have two digits after the decimal point:

```
import mlreportgen.dom.*
setDefaultNumberFormat("%0.4f");
d = Document("myDoc2","pdf");

p1 = Paragraph("pi with default format: ");
append(p1,pi);
append(d,p1);

p2 = Paragraph("pi with paragraph format: ");
p2.Style = {NumberFormat("%0.2f")};
append(p2,pi);
append(d,p2);

close(d);
rptview(d);
```

Here are the numbers in the report:

```
pi with default format: 3.1416
pi with paragraph format: 3.14
```

For an example that specifies the format for all numbers in a table, see “Format Numbers in a Table” on page 17-112.



**See Also**

`sprintf` | `mlreportgen.dom.getDefaultNumberFormat` |  
`mlreportgen.dom.setDefaultNumberFormat` | `mlreportgen.dom.Number` |  
`mlreportgen.dom.NumberFormat`

**Related Examples**

- “Format Numbers in a Table” on page 17-112

## Create and Format Paragraphs

In this section...
“Create a Paragraph” on page 13-46
“Create a Heading” on page 13-46
“Format a Paragraph” on page 13-46

### Create a Paragraph

You can create a paragraph by using an `mlreportgen.dom.Paragraph` constructor with a character vector. For example:

```
p = Paragraph('Text for a paragraph');
```

You can also specify these DOM objects in a `Paragraph` object constructor.

- `mlreportgen.dom.Text`
- `mlreportgen.dom.ExternalLink`
- `mlreportgen.dom.InternalLink`
- `mlreportgen.dom.LinkTarget`
- `mlreportgen.dom.Image`

### Create a Heading

A heading is a type of paragraph. You can use `mlreportgen.dom.Heading1`, `Heading2`, and so on, to create headings. Alternatively, you can use an `mlreportgen.dom.Heading` object if you want to use programmatically derived values for the heading level.

This example creates a first-level heading with the text `Chapter 1: System Overview`. If you create a table of contents, this heading appears at the top level.

```
h1 = Heading1('Chapter 1: System Overview');
```

### Format a Paragraph

You can format a paragraph using DOM format objects or format properties. You can also use template styles. For information about these formatting techniques and format inheritance, see “Report Formatting Approaches” on page 13-17.

---

**Note** You can use the same format objects and properties for heading objects (`Heading` and `Heading1`, `Heading2`, and so on) as you do for `Paragraph` objects.

---

#### Format a Paragraph Programmatically

You can use DOM API format objects to format `Paragraph` objects or format properties to specify commonly used paragraph formats. This example uses:

- An `OuterMargin` format object to specify the margins for the paragraph

- The HAlign format property to center the paragraph

```
import mlreportgen.dom.*;
d = Document('test','html');

p = Paragraph('Indent a half inch and space after 12 points. ');
p.Style = {OuterMargin('0.5in','0in','0in','12pt')};
append(d,p);

p = Paragraph('Centered paragraph');
p.HAlign = 'center';
append(d,p);

close(d);
rptview(d.OutputPath);
```

Use these objects and properties to format a paragraph.


Formatting	Format Object	Format Property
Font	FontFamily	FontFamilyName
Backup font (HTML only)	FontFamily	n/a
Complex script font (for example, Arabic)	FontFamily	n/a
East Asian font	FontFamily	n/a
Font size	FontSize	FontSize
Foreground color	Color	Color
Background color	BackgroundColor	BackgroundColor
Bold	Bold	Bold
Italic	Italic	Italic
Subscript or superscript	VerticalAlign	n/a
Strike through	Strike	Strike
Underline type	Underline	Underline
Underline color	Underline	n/a
Create border around paragraph	Border	n/a
Preserve white space	WhiteSpace	WhiteSpace
Indent a paragraph	OuterMargin	OuterLeftMargin
Indent first line of paragraph	FirstLineIndent	FirstLineIndent
Hanging indent	FirstLineIndent	n/a
Space before and after paragraph	OuterMargin	n/a
Space to right of paragraph	OuterMargin	n/a
Space between paragraph and its bounding box	InnerMargin	n/a
Space between paragraph lines	LineSpacing	n/a

Formatting	Format Object	Format Property
Align paragraph left, center, right	HAlign	HAlign
Start paragraph on next page	PageBreakBefore	n/a
Keep with next paragraph	KeepWithNext	n/a
Keep paragraph on same page	KeepLinesTogether	n/a
Eliminate widows and orphans	WidowOrphanControl	n/a
Table of contents level of paragraph	OutlineLevel	OutlineLevel
Display as specified	Display	n/a

### Format Paragraphs for Microsoft Word Using Template Styles

You can format a paragraph using a style in a Word template. You can add styles to the template or modify existing ones.

To add a paragraph style:

- 1 Open the Word template used with the report.
- 2 Open the **Styles** pane.
- 3 Click the **Manage Styles** button .
- 4 Click **New Style**.
- 5 In the Create New Style from Formatting dialog box, set **Style type** to Character or Linked (paragraph and character).
- 6 Format the style as needed.

For more information about working with Word styles, see “Modify Styles in a Microsoft Word Template” on page 13-126.

### Format Paragraphs Using PDF or HTML Template Styles

You can format a paragraph using a style in an HTML or PDF style sheet in your template. You can add styles to the template or modify existing ones.

Define the style using a selector on a p element. This example defines a BodyPara paragraph style.

```
p.BodyPara {
  font-family: "Times New Roman", Times, serif;
  font-style: normal;
  font-size: 11pt;
  color: black;
  margin-left: 0.5in;
}
```

You can use any CSS properties and selectors in HTML templates. For PDF templates, you can use a subset of CSS properties and selectors. See “Modify Styles in PDF Templates” on page 13-138.

For more information about using HTML styles with DOM objects, see “Modify Styles in HTML Templates” on page 13-137.

## Apply a Style to a Paragraph Object

Apply a template style to a Paragraph object either as the second argument in a Paragraph object constructor or by setting the StyleName property on the paragraph to a template style.

Suppose that you have defined styles named BodyPara and MyTitle in a template. This example first specifies a style name in a Paragraph constructor. It then specifies the style in a Paragraph object StyleName format property. This example assumes both styles are defined in MyTemplate.

```
import mlreportgen.dom.*;
rpt = Document('MyReport', 'html', 'MyTemplate');

% Specify style name using an argument when you create the Paragraph
p = Paragraph('Format this paragraph using a body style.', 'BodyPara');
append(rpt,p);

p = Paragraph('This paragraph is formatted using a title style.');
```

```
% Specify style name using a property on the paragraph
p.StyleName = 'MyTitle';
append(rpt,p);

close(rpt);
rptview(rpt.OutputPath);
```

## Override Template Formats

You can use programmatic formats to override the paragraph formats defined in a template-based paragraph style. Suppose that you define a paragraph style named BodyPara in your Word template and set the KeepWithNext property to off. You can override the style in your report program to keep a particular paragraph on the same page with the next paragraph:

```
import mlreportgen.dom.*;
rpt = Document('MyReport', 'docx', 'MyTemplate');
```

```
p = Paragraph('Keep this body paragraph with next.', 'BodyPara');
p.Style = {'KeepWithNext'};
append(rpt,p);

p = Paragraph('Next paragraph.');
```

```
append(rpt, p);

close(rpt);
rptview(rpt.OutputPath);
```

## See Also

### Classes

mlreportgen.dom.Paragraph | mlreportgen.dom.Text | mlreportgen.dom.FontFamily |  
mlreportgen.dom.FontSize | mlreportgen.dom.Bold | mlreportgen.dom.Italic |  
mlreportgen.dom.Underline | mlreportgen.dom.Strike |  
mlreportgen.dom.KeepLinesTogether | mlreportgen.dom.KeepWithNext |  
mlreportgen.dom.PageBreakBefore | mlreportgen.dom.LineSpacing |  
mlreportgen.dom.Display

### **Related Examples**

- “Add Content to a Report” on page 13-10

### **More About**

- “Report Formatting Approaches” on page 13-17

## Create and Format Lists

You can use the DOM API to create and format unordered, ordered, and multilevel lists in a report generation program.

Lists are represented as `mlreportgen.dom.UnorderedList` or `mlreportgen.dom.OrderedList` objects. List items are represented as `mlreportgen.dom.ListItem` objects.

You can create a list from a MATLAB array that specifies the list items or create a list one item at a time. Creating a list from an array is the simplest approach. Creating a list one item at a time is useful when the items contain multiple elements, such as a paragraph and a table.

Format lists and list items by using template-defined styles or programmatically by using format objects.

### Create a List from a MATLAB Array

You can create a list from a one-dimensional numeric array, string array, array of character vectors, cell array, or categorical array. Specify list items in a cell array when the items have different types.

#### Create an Unordered List from an Array

To create an unordered list from an array, append the array directly to a document or document part. The append method:

- Creates an `mlreportgen.dom.UnorderedList` object.
- Creates an `mlreportgen.dom.ListItem` object for each element of the array.
- Appends the `ListItem` objects to the `UnorderedList` object and the `UnorderedList` object to the document or document part.

For example, this code creates an unordered list from a string array:

```
import mlreportgen.dom.*
d = Document("fruit","html");

ul = append(d,["apples","oranges","bananas"]);

close(d);
rptview(d);
```

Here is the list in the generated report:

- apples
- oranges
- bananas

---

**Note** To create a list by appending an array to a document or document part, the array must be 1-by-*n*. Appending an *n*-by-1 array to a document or document part creates a table, not a list.

---

Alternatively, create an unordered list by providing an array as an input to the `mlreportgen.dom.UnorderedList` constructor. For example:

```
import mlreportgen.dom.*
d = Document("fruit", "html");

ul = UnorderedList(["apples", "oranges", "bananas"]);
ul = append(d, ul);

close(d);
rptview(d);
```

The constructor creates an `mlreportgen.dom.ListItem` object for each array element and appends the `ListItem` objects to the `UnorderedList` object.

### Create an Ordered List from an Array

To create an ordered list from an array, provide the array as input to the `mlreportgen.dom.OrderedList` constructor. The constructor creates an `mlreportgen.dom.ListItem` object for each element of the array and appends the `ListItem` objects to the `OrderedList` object. For example, this code creates an ordered list from a string array.

```
import mlreportgen.dom.*
d = Document("weekdays", "html");

ol = OrderedList(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]);
append(d, ol);

close(d);
rptview(d);
```

Here is the list in the generated report:

1. Monday
2. Tuesday
3. Wednesday
4. Thursday
5. Friday

### Create a List from a Cell Array

Specify list items in a cell array when the items have different types. In a cell array, you can include character vectors, strings, numbers, and some DOM objects, such as an `mlreportgen.dom.Text` object. For a list of DOM objects that you can include, see `mlreportgen.dom.ListItem`.

For example, this code creates an ordered list from a cell array that contains character vectors and an `mlreportgen.dom.ExternalLink` object.

```
import mlreportgen.dom.*
d = Document('myreport', 'html');

ol = UnorderedList({...
    'apples', ...
    'oranges', ...
    ExternalLink('https://en.wikipedia.org/wiki/Mango', ...
    'mango')});
append(d, ol);
```



```
close(d);
rptview(d);
```

Here is the list in the generated report:

- apples
- oranges
- [mango](#)

## Create a List One Item at a Time

Instead of specifying an entire list as an array, you can specify each list item as an `mlreportgen.dom.ListItem` object and append the `ListItem` objects to an `mlreportgen.UnorderedList` or `mlreportgen.OrderedList` object. This approach is useful when the list items contain multiple paragraphs, a mix of paragraphs and tables, or other combinations of document elements.

For example, this code creates a list in which the list items consist of a paragraph and a table.

```
import mlreportgen.dom.*
d = Document('magicsquares', 'html');

ol = UnorderedList();
item1 = ListItem(Paragraph('magic(2)'));
append(item1, Table(magic(2)));
append(ol, item1);
item2 = ListItem(Paragraph('magic(3)'));
append(item2, Table(magic(3)));
append(ol, item2);
append(d, ol);

close(d);
rptview(d);
```

Here is the list in the generated report:

- magic(2)
 

13
42
- magic(3)
 

816
357
492

## Create a Multilevel List

A multilevel list is a list that contains nested lists. You can nest any combination of ordered and unordered lists up to nine levels of nesting.

You can use the following approaches to create multilevel lists. The generated list looks the same with each approach, but the DOM API representations differ.

- Create a multilevel list from cell arrays that model the list hierarchy. With this approach, the DOM API represents a sublist as a child of the list that contains it.
- Create a multilevel list one list at a time using `mlreportgen.dom.UnorderedList`, `mlreportgen.dom.OrderedList`, and `mlreportgen.dom.ListItem` objects. Append a sublist to a list or list item. If you append the sublist to a list, the DOM API represents the sublist as a child of the list. If you append the sublist to a list item, the DOM API represents the sublist as a child of the list item.

### Create a Multilevel List from Cell Arrays

To create a multilevel list from cell arrays, use one of these approaches:

- If the sublist is an unordered list, you can represent it as a 1-by-*n* cell array that is an element of a 1-by-*n* cell array that represents the list that is one level up. For example, this code represents a sublist as a cell array that is the third element of a cell array that represents the top-level list.

```
import mlreportgen.dom.*;
d = Document('nestedListReport', 'html');

topList = OrderedList({...
    'Start MATLAB',...
    'Create a rank 3 or 4 magic square',...
    {'magic(3)',... % sublist is third element
    'magic(4)'},...
    'Close MATLAB'});
append(d, topList);

close(d);
rptview(d);
```

Here is the list in the generated report:

1. Start MATLAB
2. Create a rank 3 or 4 magic square
  - magic(3)
  - magic(4)
3. Close MATLAB

- If the sublist is an ordered list, create an `mlreportgen.dom.OrderedList` object from a 1-by-*n* cell array that represents the sublist. Include the `OrderedList` object as an element of a cell array that represents the list that is one level up. For example, this code creates an `OrderedList` object from a cell array that represents the sublist and includes the object as the third element of the cell array that is used to create the top-level list.

```
import mlreportgen.dom.*;
d = Document('orderedListReport', 'html');
```

```

topList = OrderedList({'Start MATLAB', ...
    'Create a rank 3 or 4 magic square',...
    OrderedList({... % sublist is third element
        'magic(3)',...
        'magic(4)'}),...
    'Close MATLAB'});

append(d,topList);

close(d);
rptview(d);

```

Here is the list in the generated report:

1. Start MATLAB
2. Create a rank 3 or 4 magic square
  1. magic(3)
  2. magic(4)
3. Close MATLAB

In the two previous examples, the DOM API represents the sublist as the third child of the top-level list. To access the sublist, use the `Children` property of the top-level list.

```
topList.Children(3)
```

### Create a Multilevel List One List at a Time

When sublist items contain multiple paragraphs, a mix of paragraphs and tables, or other combinations of document elements, create `mlreportgen.dom.ListItem` objects and append them to an `mlreportgen.dom.UnorderedList` or an `mlreportgen.dom.OrderedList` object. Then, append the sublist to a list or to a list item.

This example appends a sublist to the top-level list as the third list item. The items in the sublist consist of text and a table.

```

import mlreportgen.dom.*;
d = Document('orderedListReport','html');

subList = UnorderedList;
subListItem1 = ListItem('>> magic(3)');
table = append(subListItem1,Table(magic(3)));
table.Width = 'lin';
append(subList,subListItem1);
subListItem2 = ListItem('>> magic(4)');
table = append(subListItem2,Table(magic(4)));
table.Width = 'lin';
append(subList,subListItem2);

topList = OrderedList();
append(topList,ListItem('Start MATLAB'));
append(topList,ListItem('Create a rank 3 or 4 magic square'));
append(topList,subList); % sublist is item 3 of topList
append(topList,ListItem('Close MATLAB'));

```

```
append(d,topList);  
  
close(d);  
rptview(d);
```

Here is the list in the generated report:

1. Start MATLAB
2. Create a rank 3 or 4 magic square
  - o >> magic(3)  
8 1 6  
3 5 7  
4 9 2
  - o >> magic(4)  
16 2 3 13  
5 11 10 8  
9 7 6 12  
4 14 15 1
3. Close MATLAB

The DOM API represents the sublist as the third child of the top-level list. To access the sublist, use the Children property of the top-level list.

```
topList.Children(3)
```

Instead of appending a sublist to a list, you can append the sublist to a list item. The following example appends a sublist to the second list item of the top-level list. The generated list looks the same as in the previous example.

```
import mlreportgen.dom.*;  
d = Document('orderedListReport','html');  
  
topList = OrderedList({ ...  
    'Start MATLAB', ...  
    'Create a rank 3 or 4 magic square:', ...  
    'Close MATLAB'});  
  
subList = UnorderedList;  
subListItem1 = ListItem('>> magic(3)');  
table = append(subListItem1,Table(magic(3)));  
table.Width = 'lin';  
append(subList,subListItem1);  
subListItem2 = ListItem('>> magic(4)');  
table = append(subListItem2,Table(magic(4)));  
table.Width = 'lin';  
append(subList,subListItem2);  
  
% Append the sublist to the second list item  
topListItem2 = topList.Children(2);  
append(topListItem2, subList);  
  
append(d, topList);
```

```
close(d);
rptview(d);
```

The DOM API represents the sublist as the second child of the second item of the top-level list. To access the sublist, use this code:

```
topList.Children(2).Children(2)
```

## Format a List Using Template-Defined Styles

You can use list styles defined in a template to specify the indentation of each level of a list and the type of bullet or the number format used to render list items.

To use a template-defined list style to format a list, set the `StyleName` property of the list to the name of the style. For example, this code specifies that the list style name is `MyListStyle`.

```
import mlreportgen.dom.*;
d = Document('myListReport', 'html');

list = append(d, {'first item', ...
    OrderedList({'step 1', 'step 2'})}, 'second item');
list.StyleName = 'MyListStyle';

close(d);
rptview(d);
```

For Microsoft Word documents, the list style that you specify must be defined in the template that is assigned to the document.

---

**Note** A list style determines how list items are rendered regardless of the list type. If you do not specify a list style, the DOM API uses a default list style that renders the list according to type. For example, the default list style for unordered lists uses bullets to render list items. If you specify a list style for an `mlreportgen.dom.UnorderedList` object that numbers items, the items are numbered, even though the object type is unordered.

---

## Create a Word List Style

For information about creating a Word template, see “Create a Microsoft Word Template” on page 13-119.

To define a list style in a Word template:


- 1 Open the Word template file by using one of these methods:
  - In MATLAB, in the **Current Folder pane**, right-click the template file and click **Open Outside MATLAB**.
  - Outside of MATLAB, right-click the file and click **Open**.

---

**Note** Do not double-click a Word template file to open it. Double-clicking the file opens a Word document file that uses the template.

---

2

In Word, on the **Home** tab, in the **Paragraph** group, click the **Multilevel List** icon  and then click **Define New List Style**.

3 Set **Name** to your style name.

4 Specify the formatting, such as the bullet style and color for each list level.

5 Select **New documents based on this template**.6 Click **OK** to save the template.

For an example that defines a Word list style, see “Custom Styled Word List” on page 17-82.

### Create an HTML or PDF List Style

To define a list style in a cascading style sheet (CSS) for an HTML or PDF template, use the `ul` element for unordered list styles and the `ol` element for ordered list styles. You can use the child selector (`>`) to define multilevel list styles. See “Modify Styles in HTML Templates” on page 13-137 and “Modify Styles in PDF Templates” on page 13-138.

For example, this CSS code defines the appearance of a two-level unordered list that can contain ordered or unordered sublists.

```
ul.MyUnorderedList {
    list-style-type:disc;
}

ul.MyUnorderedList > ul {
    list-style-type:circle;
}

ul.MyUnorderedList > ol {
    list-style-type:decimal;
}
```

For information about editing CSS, see documentation such as the W3Schools.com CSS tutorial.

### Format a List Programmatically

Format lists programmatically by adding format objects, such as an `mreportgen.dom.Color` object, to the `Style` property of an `mreportgen.dom.UnorderedList`, `mreportgen.dom.OrderedList`, or `mreportgen.dom.ListItem` object. Formatting that you specify in the `Style` property overrides a template-defined style.

For lists in PDF and HTML reports, you can specify the bullet type or numbering type by using an `mreportgen.dom.ListStyleType` format object.

For example, this code creates a green, ordered list with lowercase alphabetic bullets. The third list item is blue and italic.

```
import mreportgen.dom.*
d = Document('myreport', 'html');

ol = OrderedList(["one", "two", "three"]);
ol.Style = {Color('Green'),ListStyleType('lower-alpha')};
ol.Children(3).Style = {Color('blue'),Italic(true)};
```

```
append(d,ol);

close(d);
rptview(d);
```

Here is the list in the generated report:

```
a. one
b. two
c. three
```

You can also format the DOM objects that you use to create list items by using the format properties or format objects associated with the object. For example, this code formats `m1reportgen.dom.Text` objects before they are used to create a list.

```
import m1reportgen.dom.*
d = Document('myreport', 'html');

li1 = Text('red');
li1.Color = 'Red';
li2 = Text('blue');
li2.Color = 'Blue';

append(d,{li1, li2});

close(d);
rptview(d);
```

Here is the list in the generated report:

```
• red
• blue
```

## Format a List Item in a Multilevel List

Depending on how you create a multilevel list, a sublist can be a child of the parent list or a child of the preceding list item in the parent list. See “Create a Multilevel List” on page 13-54.

If the sublist is a child of a list item of the parent list, the sublist inherits the formatting from the list item. For example, this code makes a sublist the child of the second list item in the parent list.

```
import m1reportgen.dom.*;
d = Document('nestedListReport', 'html');

parentlist = OrderedList();
li1 = ListItem('List Item 1');
li2 = ListItem('List Item 2');
li2.Style = [li2.Style {Color('red')});

sublist = UnorderedList({'Sublist Item 1' 'Sublist Item 2'});
append(li2, sublist);

append(parentlist, li1);
append(parentlist, li2);
append(d, parentlist);
```

```
close(d);  
rptview(d);
```

The sublist inherits the red color from the list item that contains it.

- 1. List Item 1
- 2. List Item 2
  - o Sublist Item 1
  - o Sublist Item 2

If you do not want a sublist to inherit formatting from the previous list item, format the paragraph or text in the previous list item instead of formatting the list item. For example:

```
import mlreportgen.dom.*;  
d = Document('nestedListReport', 'html');  
  
parentlist = OrderedList();  
li1 = ListItem('List Item 1');  
txt = Text('List Item 2');  
txt.Color = 'red';  
li2 = ListItem(txt);  
  
sublist = UnorderedList({'Sublist Item 1' 'Sublist Item 2'});  
  
append(parentlist, li1);  
append(parentlist, li2);  
append(parentlist, sublist);  
  
append(d, parentlist);  
  
close(d);  
rptview(d);
```

Alternatively, you can create the sublist as a child of the parent list rather than a child of the previous list item in the parent list. If the sublist is a child of the parent list, the sublist does not inherit the formatting of the previous list item in the parent list. For example, this code makes a sublist a child of the parent list:

```
import mlreportgen.dom.*;  
d = Document('nestedListReport', 'html');  
  
parentlist = OrderedList();  
li1 = ListItem('List Item 1');  
li2 = ListItem('List Item 2');  
li2.Style = [li2.Style {Color('red')});  
  
sublist = UnorderedList({'Sublist Item 1' 'Sublist Item 2'});  
  
append(parentlist, li1);  
append(parentlist, li2);  
append(parentlist, sublist);  
  
append(d, parentlist);
```



```
close(d);  
rptview(d);
```

The second list item of the parent list is red, but the sublist is black.

1. List Item 1
2. List Item 2
  - o Sublist Item 1
  - o Sublist Item 2

## See Also

[mlreportgen.dom.OrderedList](#) | [mlreportgen.dom.UnorderedList](#) |  
[mlreportgen.dom.ListItem](#) | [mlreportgen.dom.ListStyleType](#)

## More About

- “Use Style Sheet Styles” on page 13-19
- “Report Formatting Approaches” on page 13-17
- “Custom Styled Word List” on page 17-82
- “Multilevel List” on page 17-86

## Choose Type of Table to Create

You can use the DOM API and Report API to create four types of tables in your reports. The types of tables have different structures and are represented by different classes.

Type of Table	Class	Example (in PDF)	More Information																								
Formal table — A table with differently formatted header, body, and footer sections	<code>mlreportgen.dom.FormalTable</code>	<table border="1"> <thead> <tr> <th>Age</th> <th>Weight</th> <th>Height</th> </tr> </thead> <tbody> <tr> <td>38</td> <td>176</td> <td>71</td> </tr> <tr> <td>43</td> <td>163</td> <td>69</td> </tr> <tr> <td>38</td> <td>131</td> <td>64</td> </tr> </tbody> </table>	Age	Weight	Height	38	176	71	43	163	69	38	131	64	“Create Formal Tables” on page 13-66												
Age	Weight	Height																									
38	176	71																									
43	163	69																									
38	131	64																									
Informal table — A table that has only a body	<code>mlreportgen.dom.Table</code>	<table border="1"> <tbody> <tr> <td>38</td> <td>176</td> <td>71</td> </tr> <tr> <td>43</td> <td>163</td> <td>69</td> </tr> <tr> <td>38</td> <td>131</td> <td>64</td> </tr> </tbody> </table>	38	176	71	43	163	69	38	131	64	“Create Informal Tables” on page 13-64															
38	176	71																									
43	163	69																									
38	131	64																									
MATLAB table — A table constructed from a MATLAB table	<code>mlreportgen.dom.MATLABTable</code>	<table border="1"> <thead> <tr> <th>Names</th> <th>Age</th> <th>Weight</th> <th>Height</th> </tr> </thead> <tbody> <tr> <td>Sanchez</td> <td>38</td> <td>176</td> <td>71</td> </tr> <tr> <td>Johnson</td> <td>43</td> <td>163</td> <td>69</td> </tr> <tr> <td>Lee</td> <td>38</td> <td>131</td> <td>64</td> </tr> <tr> <td>Diaz</td> <td>40</td> <td>133</td> <td>67</td> </tr> <tr> <td>Brown</td> <td>49</td> <td>119</td> <td>64</td> </tr> </tbody> </table>	Names	Age	Weight	Height	Sanchez	38	176	71	Johnson	43	163	69	Lee	38	131	64	Diaz	40	133	67	Brown	49	119	64	“Create Tables from MATLAB Tables” on page 13-68
Names	Age	Weight	Height																								
Sanchez	38	176	71																								
Johnson	43	163	69																								
Lee	38	131	64																								
Diaz	40	133	67																								
Brown	49	119	64																								
Base table — A table with a numbered title	<code>mlreportgen.report.BaseTable</code>	<p><b>Table 1. Patient Data</b></p> <table border="1"> <thead> <tr> <th>Age</th> <th>Weight</th> <th>Height</th> </tr> </thead> <tbody> <tr> <td>38</td> <td>176</td> <td>71</td> </tr> <tr> <td>43</td> <td>163</td> <td>69</td> </tr> <tr> <td>38</td> <td>131</td> <td>64</td> </tr> </tbody> </table>	Age	Weight	Height	38	176	71	43	163	69	38	131	64	“Create Report API Base Tables” on page 3-34												
Age	Weight	Height																									
38	176	71																									
43	163	69																									
38	131	64																									

The following guidelines help you choose the type of table to create based on your table requirements. The guidelines are not exhaustive. In some cases, more than one type of table might meet your requirements. In these cases, you can choose the type of table based on your preferences.

Requirement	Type of Table
Create a table with a header or footer.	<ul style="list-style-type: none"> <li>• Formal table, if the header or footer has multiple rows.  The separate header, body, and footer sections facilitate adding content to and formatting the header, body, or footer.</li> <li>• Informal or formal table, if the header has one row.  You can format the first row of an informal header to look like a header.</li> </ul>
Create a table without a header or footer.	Informal table
Convert a MATLAB table to a DOM table.	DOM representation of a MATLAB table ( <code>mlreportgen.dom.MATLABTable</code> )
Create a table with a numbered title.	Base table  You can first create a DOM table, and then create the base table from the DOM table or you can create the base table directly from a MATLAB array.
Divide a wide table into legible slices.	Base table  If a DOM table is too wide, you can create a base table from the DOM table and then use the <code>BaseTable</code> properties for slicing the table. You can also slice a table created as a <code>Table</code> or <code>FormalTable</code> object by using <code>mlreportgen.utils.TableSlicer</code> .

## See Also

`mlreportgen.dom.Table` | `mlreportgen.dom.FormalTable` | `mlreportgen.report.BaseTable` | `mlreportgen.dom.MATLABTable`

## More About

- “Create Informal Tables” on page 13-64
- “Create Formal Tables” on page 13-66
- “Create Tables from MATLAB Tables” on page 13-68
- “Create Report API Base Tables” on page 3-34

## Create Informal Tables

A DOM API informal table has only a body. It does not have separate header or footer sections. By contrast, a DOM API formal table has separate header, body, and footer sections. You can format one or more rows of an informal table to look like a header or footer. However, if your table header or footer has multiple rows, consider using a formal table. See “Create Formal Tables” on page 13-66. For information about other types of tables that you can create in reports, see “Choose Type of Table to Create” on page 13-62.

An informal table is represented by an `mlreportgen.dom.Table` object. Table rows and entries are represented by `mlreportgen.dom.TableRow` and `mlreportgen.dom.TableEntry` objects, respectively.

You can create an informal table from a MATLAB array. This approach is the simplest because you do not have to create `TableRow` and `TableEntry` objects. You can also create an informal table by building the table from `TableRow` and `TableEntry` objects. This approach is useful for complex formatting requirements, such as when an entry spans multiple columns or rows.

For information about formatting tables, see “Format Tables” on page 13-72.

### Create an Informal Table from a MATLAB Array

You can create a table by appending a two-dimensional numeric array, categorical array, or cell array to a document. A cell array can contain MATLAB data and DOM objects. See the `array` input argument on the `mlreportgen.dom.Table` reference page. The `append` method converts an array to an `mlreportgen.dom.Table` object, appends it to the document, and returns the `Table` object, which you can then format. You can also create a `Table` object directly by providing the array to the `Table` constructor.

This example creates a table from a numeric array. The example uses `Table` format properties to specify the table border, the row and column separators, and the alignment and margin of the entries.

```
import mlreportgen.dom.*;
d = Document('myTableReport', 'pdf');

t = append(d, magic(5));
t.Border = 'single';
t.ColSep = 'single';
t.RowSep = 'single';
t.TableEntriesHAlign = 'right';
t.TableEntriesVAlign = 'middle';
t.TableEntriesInnerMargin = '2pt';

close(d);
rptview(d);
```

Here is the table in the generated report.

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

## Build an Informal Table from Rows and Entries

You can create an informal table by creating `mlreportgen.dom.TableEntry` objects, appending them to `mlreportgen.dom.TableRow` objects, and appending the `mlreportgen.dom.TableRow` objects to an `mlreportgen.dom.Table` object. For an example, see “Span a Table Entry Across Rows and Columns” on page 17-13.

### See Also

`mlreportgen.dom.Table` | `mlreportgen.dom.TableRow` | `mlreportgen.dom.TableEntry`

### More About

- “Choose Type of Table to Create” on page 13-62
- “Format Tables” on page 13-72
- “Report Formatting Approaches” on page 13-17

## Create Formal Tables

A DOM API formal table has header, body, and footer sections. A formal table is useful when a header or footer has multiple rows. The separate header, body, and footer sections facilitate adding rows to and formatting the header, body, or footer independently of the other sections. For information about other types of tables that you can create in reports, see “Choose Type of Table to Create” on page 13-62. For information about formatting tables, see “Format Tables” on page 13-72.

A formal table is represented by an `mlreportgen.dom.FormalTable` object. The header, body, and footer sections are represented by `mlreportgen.dom.TableHeader`, `mlreportgen.dom.TableBody`, and `mlreportgen.dom.TableFooter` objects, respectively. Each section has rows and table entries. The rows are represented by `mlreportgen.dom.TableRow` objects. Entries in the body and footer sections are represented by `mlreportgen.dom.TableEntry` objects. Entries in the header are represented by `mlreportgen.dom.TableHeaderEntry` or `mlreportgen.dom.TableEntry` objects.

You can create the sections of a formal table from a MATLAB array. This approach is the simplest because you do not have to create `TableRow` and `TableEntry` or `TableHeaderEntry` objects. You can also create the sections by building the table from `TableRow` and `TableEntry` or `TableHeaderEntry` objects. This approach is useful for complex formatting requirements, such as when an entry spans multiple columns or rows. You can combine the approaches. For example, you can create the table body from an array and build the header from rows and entries.

### Create a Formal Table from MATLAB Arrays

Create a formal table by using the `mlreportgen.dom.FormalTable` constructor. The constructor optionally accepts a two-dimensional array or a cell array of MATLAB data for the body, header, and footer sections. See the header, body, and footer arguments on the `mlreportgen.dom.FormalTable` reference page. For an example that creates a formal table header from a string array and a formal table body from a cell array, see “Create a Table from a Cell Array” on page 17-109.

### Create a Formal Table from Rows and Entries

You can build the header, body, or footer sections of a formal table by appending table entries to table rows and appending the rows to the sections.

To create a header entry, you can use an `mlreportgen.dom.TableHeaderEntry` or `mlreportgen.dom.TableEntry` object. If you create a header from an array, the DOM API creates the entries as `TableHeaderEntry` objects. In an HTML report, the DOM API renders a `TableHeaderEntry` object as a `th` (table header cell) element. This rendering relies on the browser to format the table entry. For Word and PDF reports, by default, the DOM API renders the `TableHeaderEntry` object as an unformatted Word or PDF table entry element. You can use the `TableHeaderEntry` object properties to format the table entries generated for all output types. For example, you can use these properties to override browser formatting in HTML reports.

To build a section of a formal table from table rows and entries, you can use the `FormalTable.appendHeaderRow`, `append`, and `appendFooterRow` methods to append rows to the table header, body, or footer sections. Alternatively, you can access a section by using the `Header`, `Body`, or `Footer` properties of the `FormalTable` object and then append rows to the section by using the `append` method.

For an example that builds a header from entries and rows, see Formal Table on page 17-0 in “Span a Table Entry Across Rows and Columns” on page 17-13.

## See Also

`mlreportgen.dom.FormalTable` | `mlreportgen.dom.TableEntry` |  
`mlreportgen.dom.TableBody` | `mlreportgen.dom.TableFooter` |  
`mlreportgen.dom.TableHeader` | `mlreportgen.dom.TableHeaderEntry` | `appendFooterRow`  
| `appendHeaderRow`

## More About

- “Choose Type of Table to Create” on page 13-62
- “Report Formatting Approaches” on page 13-17
- “Create a Table from a Cell Array” on page 17-109
- “Format Tables” on page 13-72

## Create Tables from MATLAB Tables

To create a DOM API table from a MATLAB table, use an `mlreportgen.dom.MATLABTable` object. The DOM representation of a MATLAB table has the structure of a DOM formal table. See “Create Formal Tables” on page 13-66. The `MATLABTable` header contains the column names from the MATLAB table. The `MATLABTable` body contains the rows and elements from the MATLAB table.

If the table in MATLAB has row names, the first column of the `MATLABTable` object contains the row names. In MATLAB, the row name column does not have a label. In the `MATLABTable` object, you can specify a label for the row name column by changing the content of the first element of the header. You can specify a line under the row name column label by setting the `RowNamesRule` property of the `MATLABTable` object to `true`.

For information about other types of tables that you can create in reports, see “Choose Type of Table to Create” on page 13-62.

### Create a Table from a MATLAB Table

This example creates a DOM table from a MATLAB table that has row names. The example generates two reports that include the DOM table. In the first report, the row names column of the table does not have a label. In the second report, the row names column has a label.

Create a MATLAB table that shows the age, weight, and height of patients. Use the `RowNames` option to identify each row by the last name of the patient.

```
LastName = {'Sanchez'; 'Johnson'; 'Lee'; 'Diaz'; 'Brown'};
Age = [38; 43; 38; 40; 49];
Weight = [176; 163; 131; 133; 119];
Height = [71; 69; 64; 67; 64];
mltable = table(Age, Weight, Height, 'RowNames', LastName)
```

```
mltable=5x3 table
      Age      Weight      Height
      ---      ---      ---
Sanchez    38         176         71
Johnson   43         163         69
Lee        38         131         64
Diaz       40         133         67
Brown     49         119         64
```

The MATLAB table has five rows and three columns. The row names are not part of the table. They are stored in a property of the table.

Create an `mlreportgen.dom.MATLABTable` object from the MATLAB table.

```
import mlreportgen.dom.*
mltableObj = MATLABTable(mltable);
```

Create a document and append the `MATLABTable` object to the document. Close and view the document.

```
d = Document('MyMATLABTable1', 'docx');
append(d, mltableObj);
```



```
close(d);
rptview(d);
```

Here is the table in the generated report:

	<u>Age</u>	<u>Weight</u>	<u>Height</u>
<b>Sanchez</b>	38	176	71
<b>Johnson</b>	43	163	69
<b>Lee</b>	38	131	64
<b>Diaz</b>	40	133	67
<b>Brown</b>	49	119	64

The DOM table is a formal table, which has a header and a body. The table body has five rows and four columns. The first column consists of the MATLAB table row names.

Generate the report again, this time with a label for the column of row names. To specify the label, replace the empty text in the first entry of the table header row with the label text. To draw a line under the label, set the `RowNamesRule` property of the `MATLABTable` object to `true`.

```
LastName = {'Sanchez'; 'Johnson'; 'Lee'; 'Diaz'; 'Brown'};
Age = [38;43;38;40;49];
Weight = [176;163;131;133;119];
Height = [71;69;64;67;64];
mltable = table(Age,Weight,Height,'RowNames',LastName);
import mlreportgen.dom.*
mltableObj = MATLABTable(mltable);
th = mltableObj.Header;
thentry11 = entry(th,1,1);
thentry11.Children(1).Children(1).Content = 'Names';
mltableObj.RowNamesRule = true;
d = Document('MyMATLABTable2', 'docx');
append(d,mltableObj);
close(d);
rptview(d);
```

Here is the table in the generated report:

<u>Names</u>	<u>Age</u>	<u>Weight</u>	<u>Height</u>
<b>Sanchez</b>	38	176	71
<b>Johnson</b>	43	163	69
<b>Lee</b>	38	131	64
<b>Diaz</b>	40	133	67
<b>Brown</b>	49	119	64

## Format a Table Created from a MATLAB Table

By default, a table generated from a `MATLABTable` object is formatted to look like a table in MATLAB. To customize the appearance of the table, you can use the same approaches that you use for other types of tables:

- Update the default style in the style sheets of the default HTML, Word, or PDF template. The default `StyleName` of a `MATLABTable` object is "rgMATLABTable".

- Set the `StyleName` property to a custom style.
- Use format properties or format objects with the table or a section of the table.

See “Format Tables” on page 13-72.

### Format the Sections of a MATLAB Table

This example shows how to format the sections of a DOM table that is created from a MATLAB table.

#### Format the Table Header

This example makes the column headings italic.

<i>Age</i>	<i>Weight</i>	<i>Height</i>
38	176	71
43	163	69
38	131	64
40	133	67
49	119	64

To access the header, use the `Header` property of the `mlreportgen.dom.MATLABTable` object. The example adds an `mlreportgen.dom.Italic` format object to the `Style` property of the `mlreportgen.dom.TableHeader` object that represents the header.

```
import mlreportgen.dom.*
d = Document('myMATLABTable', 'pdf');

Age = [38;43;38;40;49];
Height = [71;69;64;67;64];
Weight = [176;163;131;133;119];
mltable = table(Age,Weight,Height);

mltableObj = MATLABTable(mltable);
mltableObj.Header.Style = [mltableObj.Header.Style {Italic(true)}];

append(d,mltableObj);
close(d);
rptview(d);
```

#### Format the Table Body

This example makes the entries of the first row of the table body blue.

<i>Age</i>	<i>Weight</i>	<i>Height</i>
38	176	71
43	163	69
38	131	64
40	133	67
49	119	64

To access the table body, use the `Body` property of the `mlreportgen.dom.MATLABTable` object. To access a row of the body, use the `row` method of the `mlreportgen.dom.TableRow` object that

represents the row. The example adds an `mlreportgen.dom.Color` format object to the `Style` property of the `TableRow` object that represents the first row.

```
import mlreportgen.dom.*
d = Document('myMATLABTable','pdf');

Age = [38;43;38;40;49];
Height = [71;69;64;67;64];
Weight = [176;163;131;133;119];
mltable = table(Age,Weight,Height);

mltableObj = MATLABTable(mltable);
tbody = mltableObj.Body;
row1 = row(tbody,1);
row1.Style = [row1.Style {Color('blue')}]];

append(d,mltableObj);
close(d);
rptview(d);
```

## See Also

`mlreportgen.dom.MATLABTable`

## More About

- “Create Tables and Assign Data to Them”
- “Choose Type of Table to Create” on page 13-62
- “Format Tables” on page 13-72

## Format Tables

You can use `mlreportgen.dom.Table`, `mlreportgen.dom.FormatTable`, `mlreportgen.dom.MATLABTable`, or `mlreportgen.report.BaseTable` objects to create a table in a report. See “Choose Type of Table to Create” on page 13-62. You can format any of these types of tables or the elements (section, row, column group, or entry) of the tables by using these approaches:

- Format the content before you create a table. For example, format numbers in MATLAB before you use them to create a table. See “Format Numbers in a Table” on page 17-112.
- Format DOM objects before you create a table from them. For example, format an `mlreportgen.dom.Paragraph` object before you use it to create a table entry.
- Modify the default template style or create a custom template style for a table or table element. See “Format Tables Using Template-Defined Styles” on page 13-72.
- Override template styles by using format properties and format objects with the object that represents a table or table element. See “Format Tables Programmatically” on page 13-75.

For information specific to formatting a `MATLABTable` table, see “Create Tables from MATLAB Tables” on page 13-68. For information specific to formatting a `BaseTable` table, see “Create Report API Base Tables” on page 3-34.

For a table object, table element object, or object contained in a table element object, a format specified by a format property or format object overrides the equivalent format specified by a template-based style. A format specified for an object overrides the equivalent format specified by the container object. For example, if a table entry contains an `mlreportgen.dom.Paragraph` object, the text color specified for the `Paragraph` object overrides the color specified for the row that contains it. The row color overrides the color specified for the table that contains the row. See “Format Inheritance” on page 13-21.

### Format Tables Using Template-Defined Styles

Tables and table elements have a default, template-defined style. Consider customizing a template-defined style if the customization applies to multiple tables and you are comfortable working with styles in Microsoft Word or editing Cascading Style Sheets (CSS) for HTML or PDF reports.

#### Create a Word Table Style

You can format a table by applying a custom Microsoft Word style to the table. If you apply a table style to one or more sections of a Word formal table, specify the widths of each of the table columns. Otherwise, the columns of the sections might not line up.



To define a table style in a Word style sheet:

- 1 Create a Microsoft Word template. For information about creating a Microsoft Word template, see “Create a Microsoft Word Template” on page 13-119.
- 2 Open the Word template file by using one of these methods:
  - In MATLAB, in the **Current Folder pane**, right-click the template file and click **Open Outside MATLAB**.
  - Outside of MATLAB, right-click the file and click **Open**.

---

**Note** Do not double-click a Word template file to open it. Double-clicking the file opens a Word document file that uses the template.

---

- 3 In Word, on the **Home** tab, in the **Styles** group, click the **Styles** icon .
- 4 Click the **Manage Styles** button .
- 5 Click **New Style**.
- 6 In the Create New Style from Formatting dialog box:
  - Specify the **Name**.
  - Set **Style type** to Table.
  - In **Style based on**, select the base style for your new style.
  - In the **Formatting** section, specify the formatting and which parts of the table the formatting applies to.
- 7 Select **New documents based on this template** and then click **OK**.
- 8 In the **Manage Styles** dialog box, select **New documents based on this template** and then click **OK**.
- 9 Save the template.

For an example of formatting using a Word template, see “Create a Zebra-Striped Table” on page 17-54.

### Create an HTML or PDF Table Style

You can format HTML and PDF tables by using a CSS style defined in a template. To create an HTML or PDF template, see “Create an HTML or PDF Template” on page 13-130.

To define a table style in an HTML or PDF template, use a `table` selector with a class name. For example, this CSS code specifies the style for tables with class `MyTable`.

```
table.MyTable {
  border-style: solid;
  border-bottom-color: rgb(128, 128, 128);
  border-bottom-width: thin;
  border-collapse: collapse;
}
```

You can use the CSS descendant selector (space) or child selector (>) to specify the format of descendants or children of a table. For example, this CSS code specifies the format of the table entries (`td` elements) of a table whose style is `MyTable`.

```
table.MyTable td {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 11pt;
  text-align: center;
}
```

See “Modify Styles in HTML Templates” on page 13-137 and “Modify Styles in PDF Templates” on page 13-138.

For information about editing CSS, see documentation such as <https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>.

For an example of formatting using an HTML template, see “Create a Zebra-Striped Table” on page 17-54.

### Apply a Style to a Table or Table Element

Once you have defined a style in a template, you can apply it to the object that represents a table or table element in your report program. Provide the style as an argument to the object constructor or assign it to the `StyleName` property of the object. You can apply a style to the header, body, or footer section of an `mlreportgen.dom.FormatTable` or `mlreportgen.dom.MATLABTable` object by assigning the style to the `StyleName` property of the `mlreportgen.dom.TableHeader`, `mlreportgen.dom.TableBody`, or `mlreportgen.dom.TableEntry` object.

For example, suppose that you defined styles named `BodyPara`, `TableTitle`, and `RuledTable` in the template for your report. This example specifies style names in a `Paragraph` constructor, in the `StyleName` property of a `Paragraph` object, and in a `Table` constructor.

```
import mlreportgen.dom.*;
rank = 5;
rpt = Document('MyReport', 'html', 'MyTemplate');

p = Paragraph('Here is a magic square of rank 5:', 'BodyPara');
append(rpt, p);

p = Paragraph(sprintf('Rank %d MagicSquare', rank));
p.StyleName = 'TableTitle';

append(rpt, Table(magic(rank), 'RuledTable'));

close(rpt);
rptview(rpt.OutputPath);
```

You can use programmatic formats to override the styles defined in a template-based table style. For example, suppose that you define a table style named `UnruledTable` in your template to create tables without borders or column or row separators. You can then override the style in your report program to draw a frame around a table.

```
import mlreportgen.dom.*;
rpt = Document('MyReport', 'html', 'MyTemplate');

table = Table(magic(5), 'UnruledTable');
table.Border = 'single';
append(rpt, table);

close(rpt);
rptview(rpt.OutputPath);
```

For more information about programmatic formatting, see “Format Tables Programmatically” on page 13-75.

### Format Table Entries Using Style Sheets

For HTML and PDF reports, you can use styles defined in an HTML template style sheet to format table entries. When defining a table entry style, use a `td` element selector. For example:

```
td.TableEntryWithBorder {
    border:5px solid red;
}
```

To apply a template-defined style to a table entry, set the `TableEntry` object `StyleName` property to the name of the style or specify the style name as the second argument to the `TableEntry` constructor. For example:

```
te = TableEntry('Hello World', 'TableEntryWithBorder');
```

## Format Tables Programmatically

If you are not comfortable editing CSS or you want to override the default table style for only a few tables or table elements, you can format tables and table elements programmatically. Use one of these approaches:

- Set the format properties of the object that represents the table or table element.
- Add format objects to the `Style` property of the object that represents the table or table element. Add format objects by concatenating the existing value of the `Style` property with a cell array that contains the new format objects. For example:

```
table.Style = [table.Style {Border('solid', 'black', '3px')}];
```

Here are some of the format objects and corresponding format properties that apply to `mlreportgen.dom.Table`, `mlreportgen.dom.FormatTable`, and `mlreportgen.dom.MATLABTable` objects.

Formatting	Format Object	Format Property
Width of table	<code>mlreportgen.dom.Width</code>	<code>Width</code>
Color of table background	<code>mlreportgen.dom.BackgroundColor</code>	<code>BackgroundColor</code>
Specify border around table	<code>mlreportgen.dom.Border</code>	<code>Border</code>
Color of border	<code>mlreportgen.dom.Border</code>	<code>BorderColor</code>
Thickness of border	<code>mlreportgen.dom.Border</code>	<code>BorderWidth</code>
Specify left, right, top, or bottom table border	<code>mlreportgen.dom.Border</code>	n/a
Collapse table and table entry borders (HTML reports only)	<code>mlreportgen.dom.BorderCollapse</code>	<code>BorderCollapse</code>
Specify column separator	<code>mlreportgen.dom.ColSep</code>	<code>ColSep</code>
Column separator color	<code>mlreportgen.dom.ColSep</code>	<code>ColSepColor</code>
Column separator thickness	<code>mlreportgen.dom.ColSep</code>	<code>ColSepWidth</code>
Specify row separator	<code>mlreportgen.dom.RowSep</code>	<code>RowSep</code>
Row separator color	<code>mlreportgen.dom.RowSep</code>	<code>RowSepColor</code>
Row separator thickness	<code>mlreportgen.dom.RowSep</code>	<code>RowSepWidth</code>
Indent table from left margin	<code>mlreportgen.dom.OuterMargin</code>	<code>OuterLeftMargin</code>
Space before or after table	<code>mlreportgen.dom.OuterMargin</code>	n/a
Space to right of table	<code>mlreportgen.dom.OuterMargin</code>	n/a

Formatting	Format Object	Format Property
Align table left, right, or center	<code>mlreportgen.dom.HAlign</code>	<code>HAlign</code>
Specify table entry flow direction (left-to-right or right-to-left)	<code>mlreportgen.dom.FlowDirection</code>	<code>FlowDirection</code>
Resize table columns to fit contents	<code>mlreportgen.dom.ResizeToFitContents</code>	n/a

You can use other format objects that apply to the objects that the table contains. The DOM API ignores format objects that do not apply to a particular object.

### Format an Informal Table Programmatically

For formatting that applies to an entire table, use format properties and objects with the object that represents the entire table. This example uses format objects to specify the table border and the row and column separators. It uses a format property to specify the background color.

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

```
import mlreportgen.dom.*
d = Document('test', 'html');

table = Table(magic(5));
table.Style = [table.Style {Border('inset', 'red', '3px'), ...
    ColSep('single', 'black', '1px'), ...
    RowSep('single', 'black', '1px')}]};

table.BackgroundColor = 'lightsteelblue';

append(d, table);

close(d);
rptview(d.OutputPath);
```

### Format a Formal or MATLAB Table Programmatically

You can use format properties and objects to format `mlreportgen.dom.FormalTable` and `mlreportgen.dom.MATLABTable` objects. If you specify a format for the table and one of its sections, the value you specify for the section overrides the value for the table as a whole. Not all formal table formats apply to formal table sections. For example, the `OuterLeftMargin` property does not apply to formal table sections. You cannot indent a header, body, or footer section independently of the table that contains it.

### Format a Table Entry

To access a table entry for formatting, you can use the `entry` method of an `mlreportgen.dom.Table`, `mlreportgen.dom.TableHeader`, `mlreportgen.dom.TableBody`, or `mlreportgen.dom.TableFooter` object.



The entry method returns an `mlreportgen.dom.TableEntry` or `mlreportgen.dom.TableHeaderEntry` object. To format the entry, you can use these `TableEntry` format properties or objects.

Formatting	Format Object	Format Property
Create border around entry	<code>mlreportgen.dom.Border</code>	<code>Border</code>
Color of border	<code>mlreportgen.dom.Border</code>	<code>BorderColor</code>
Thickness of border	<code>mlreportgen.dom.Border</code>	<code>BorderWidth</code>
Create left, right, top, or bottom entry border	<code>mlreportgen.dom.Border</code>	n/a
Align entry content to top, bottom, or middle	<code>mlreportgen.dom.VAlign</code>	<code>VAlign</code>
Space between entry boundary and entry content	<code>mlreportgen.dom.InnerMargin</code>	<code>InnerMargin</code>
Space between entry content and its top, bottom, right, or left boundaries	<code>mlreportgen.dom.InnerMargin</code>	n/a
Cause entries to span multiple columns	n/a	<code>ColSpan</code>
Cause entry to span multiple rows	n/a	<code>RowSpan</code>

You can use other format objects that apply to the objects that the table entry contains. The DOM API ignores format objects that do not apply to a particular object.

This example creates a table from the output of the `magic` function and makes the maximum number of the table red.

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

```
import mlreportgen.dom.*;

d = Document('test', 'pdf');

m = magic(5);
[v,i] = max(m);
[v1,i1] = max(max(m));
t = Table(m);
t.Border = 'single';
t.ColSep = 'single';
t.RowSep = 'single';

t.TableEntriesInnerMargin = '2pt';
t.TableEntriesHAlign = 'right';
```

```
maxnum = entry(t,i(i1),i1);
maxnum.Children(1).Color = 'Red';

append(d,t);

close(d);
rptview(d);
```

### Format All Table Entries

To specify the same format or set of formats for all of the entries of a DOM table or a section of a DOM table, you can use these properties:

- TableEntriesValign
- TableEntriesHalign
- TableEntriesInnerMargin
- TableEntriesStyle

This example creates a table from a MATLAB table. The example:

- Makes all table body entries blue by adding a format object to the TableEntriesStyle property
- Centers all of the table body entries by setting the TableEntriesHalign property

#### Age Weight Height

Age	Weight	Height
38	176	71
43	163	69
38	131	64
40	133	67
49	119	64

```
import mlreportgen.dom.*
d = Document('outermargin','docx');

import mlreportgen.dom.*
d = Document('myMATLABTable','pdf');

Age = [38;43;38;40;49];
Height = [71;69;64;67;64];
Weight = [176;163;131;133;119];
mltable = table(Age,Weight,Height);

mltableObj = MATLABTable(mltable);
tbodyObj = mltableObj.Body;
tbodyObj.TableEntriesStyle = {Color('blue')};
tbodyObj.TableEntriesHalign = 'center';

append(d,mltableObj);

close(d);
rptview(d);
```

## Format a Table Row

To access a table row for formatting, you can use the row method of an `mlreportgen.dom.Table`, `mlreportgen.dom.TableHeader`, `mlreportgen.dom.TableBody`, or `mlreportgen.dom.TableFooter` object.

The row method returns an `mlreportgen.dom.TableRow` object. To format the row, you can use these `TableRow` format properties or objects.

Formatting	Format Object	Format Property
Specify the exact height of a row	<code>mlreportgen.dom.RowHeight</code>	Height
Specify the minimum height of a row (Word reports only)	<code>mlreportgen.dom.RowHeight</code>	n/a
Cause this row to repeat as header row when a table flows across pages	<code>mlreportgen.dom.RepeatAsHeaderRow</code>	n/a
Allow this row to straddle a page boundary	<code>mlreportgen.dom.AllowBreakAcrossPages</code>	n/a

You can use other format objects that apply to the objects that the row contains. The DOM API ignores format objects that do not apply to a particular object.

This example creates a table from the output of the magic function and makes the content of the first row red.

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

```
import mlreportgen.dom.*;

d = Document('test', 'pdf');

m = magic(5);
[v, i] = max(m);
[v1, i1] = max(max(m));
t = Table(m);
t.Border = 'single';
t.ColSep = 'single';
t.RowSep = 'single';

t.TableEntriesInnerMargin = '2pt';
t.TableEntriesHAlign = 'right';

r = row(t, 1);
r.Style = [r.Style {Color('red')});
append(d, t);
close(d);
rptview(d);
```

### Format Table Columns

To specify the format of a group of adjacent table columns, use an `mlreportgen.dom.TableColSpecGroup` object. To override the formats of a column group for some columns, use an `mlreportgen.dom.TableColSpec` object.

In this example, the `TableColSpecGroup` object specifies green text. The `TableColSpec` object overrides the formats for the first column, specifying bold, red text.

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

```
import mlreportgen.dom.*
rpt = Document('test', 'pdf');

rank = 5;
t = Table(magic(rank));
t.Border = 'single';
t.ColSep = 'single';
t.RowSep = 'single';
t.TableEntriesInnerMargin = '2pt';
t.TableEntriesHAlign = 'right';

%Specify the formats for all columns
grps(1) = TableColSpecGroup;
grps(1).Span = rank;
grps(1).Style = {Color('green')};

%Specify the formats for the first column
specs(1) = TableColSpec;
specs(1).Span = 1;
specs(1).Style = {Bold(true),Color('red')};

grps(1).ColSpecs = specs;

t.ColSpecGroups = grps;
append(rpt,t);

close(rpt);
rptview(rpt.OutputPath);
```

To resize columns to fit the widest content of the table entries in a column, include a `ResizeToFitContents` object in the `Style` property of the table.

### See Also

`mlreportgen.dom.Table` | `mlreportgen.dom.TableRow` | `mlreportgen.dom.TableEntry` | `mlreportgen.dom.FormatTable` | `mlreportgen.dom.MATLABTable` | `mlreportgen.report.BaseTable` | `mlreportgen.dom.TableHeaderEntry` | `mlreportgen.dom.TableColSpecGroup` | `mlreportgen.dom.TableColSpec`

**More About**

- “Choose Type of Table to Create” on page 13-62
- “Report Formatting Approaches” on page 13-17

**External Websites**

- <https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>

## Create Links

### In this section...

“Create an Internal Link” on page 13-82  
 “Create an External Link” on page 13-82  
 “Add Text or Images to Links” on page 13-83  
 “Create a Page Reference” on page 13-83

You can add these kinds of links to a report:

- Internal — Link to locations in the report. Use an `mreportgen.dom.InternalLink` object.
- External — Link to a location outside of the report, such as an HTML page or a PDF file. Use an `mreportgen.dom.ExternalLink` object.

### Create an Internal Link

To link from one location in a document to another location in the same document:

- 1 Define the link target by appending an `mreportgen.dom.LinkTarget` object to the document or document element. To make sure that the link target name that you use to create the `LinkTarget` object is valid for all report types, generate the link target name by using `mreportgen.utils.normalizeLinkID`.
- 2 Define the link by appending an `mreportgen.dom.InternalLink` object to the document or document element. When you create the `InternalLink` object, for the `targetName` argument, use the link target name from the `LinkTarget` object.

For example, you can include an `About the Author` link to a heading with the title `Author's Biography`.

```
import mreportgen.dom.*
import mreportgen.utils.*
d = Document('mydoc', 'pdf');

% Append a link target to a heading
h = Heading(1, 'Author's Biography');
h.Style = {PageBreakBefore(true)};
linkID = normalizeLinkID('bio');
append(h, LinkTarget(linkID));

% Link to the target
append(d, InternalLink(linkID, 'About the Author'));

% Append the heading
append(d, h);

close(d);
rptview(d);
```

### Create an External Link

Use an `mreportgen.dom.ExternalLink` object to create an external link, specifying the link target and the link text.

```
import mlreportgen.dom.*
d = Document('mydoc');

append(d,ExternalLink('https://www.mathworks.com/', 'MathWorks'));

close(d);
rptview('mydoc', 'html');
```

## Add Text or Images to Links

To add text or an image to an ExternalLink or InternalLink object, use the append method with that object. Append a Text, Image, or CustomElement object.

## Create a Page Reference

You can create a numeric reference to the page where a link target resides. For example, you can create a page reference in the form “See page 15,” where the target you are referencing is on an object on page 15. For example:

```
import mlreportgen.dom.*;
d = Document('mydoc', 'pdf');
open(d);

% Add target to heading object and append heading and
% para text to document
h = Heading1(LinkTarget('mytarget'));
append(h, 'Referenced Head');
p = Paragraph('Here is some paragraph text. ');
append(d, h);
append(d, p);

% Add another page and insert the page reference
% to the target
p1 = Paragraph('The following paragraph contains the page reference. ');
p1.Style = {PageBreakBefore(true)};
p2 = Paragraph('See Page ');
p2.WhiteSpace = 'preserve';
ref = PageRef('mytarget');
append(p2, ref);
append(p2, '. ');
append(d, p1);
append(d, p2);

close(d);
rptview(d.OutputPath);
```

In your PDF template, you can use a <pageref> element to create this kind of reference. Your DOM API program must set the link target that the element uses. The <pageref> uses one argument: <pageref target="nameoftarget">.

For more information on this mechanism, see mlreportgen.dom.PageRef.

## See Also

mlreportgen.dom.ExternalLink | mlreportgen.dom.InternalLink |  
mlreportgen.dom.LinkTarget | mlreportgen.dom.PageRef | append

### **Related Examples**

- “Create Image Maps” on page 13-99
- “Add Content to a Report” on page 13-10

### **More About**

- “Report Formatting Approaches” on page 13-17



## Create a Dynamic Table

A dynamic table is one whose size you do not know before your report generator program runs, so you cannot hard code its size. This example shows two approaches to creating a dynamic table. One approach creates a table from basic table objects. The other approach uses a table constructor that creates a table directly from the input to the constructor.

### Create Dynamic Table From Table Objects

This program shows how to create a table by looping and creating basic table objects: table, table entry, and table row objects. The code displays a table of test results, with the first column being the test name, the second, the test time, and the third, the color-coded result.

Name	Time	Result
Test 1	1.25	Pass
Test 2	1.43	Fail
Test 3	1.51	Pass
Test 4	2.17	Fail

The code first determines the table header row text and the number of table columns from the data in a struct. It then creates a formal table object and specifies the table formatting. The program then begins building the table by looping through the heading text items, creating table entries, and adding the table entries to create the table heading row. Then, the code loops through the data and creates a table row and table entries. It builds the table by adding each table entry to its table row, and then adds each table row to the table.

```
import mlreportgen.report.*
import mlreportgen.dom.*

rpt = Report('testResults','pdf');

% Input data
testData = struct('Name',{'Test 1','Test 2',...
    'Test 3','Test 4'},'Time',{1.25,1.43,1.51,2.17},...
    'Result',{'Pass','Fail','Pass','Fail'});

% Row heading names and number of columns
fields = fieldnames(testData);
nFields = numel(fields);

% Table, row, and table entries formatting
table = FormalTable();
table.RowSep = 'Solid';
table.ColSep = 'Solid';
table.Border = 'Solid';
table.TableEntriesInnerMargin = '5px';
table.Header.Style = {Bold()};
```

```

% Table heading row
headRow = TableRow();
for k = 1:nFields
    append(headRow,TableEntry(fields{k}));
end
append(table.Header,headRow);

% Table rows and table entries
for data = testData
    row = TableRow();
    for j = 1:nFields
        x = string(data.(fields{j}));
        p = Paragraph(x);
        if x == "Pass"
            p.Color = 'green';
        elseif x == "Fail"
            p.Color = 'red';
        end
        new_entry = TableEntry(p);
        append(row,new_entry);
    end
    append(table,row);
end
add(rpt,table);

close(rpt);
rptview(rpt);

```

## Create Dynamic Table Using Table Constructor

This program shows how to create a table using the input to the table constructor. The advantage of creating a table this way is that you do not have to build the table by looping through the data to create table entry and row objects. In this example, the input data that specifies the table content is in a cell array. This code displays a the same table of test results as shown in “Create Dynamic Table From Table Objects” on page 13-85.

The code first determines the number of rows and columns in the cell array and preallocates memory for the table. The code then performs two optional steps that format the table contents — converting the data values to strings to specify the number of decimal places for the data, and looping through the data to set the color of the result column. Finally, it creates a formal table directly from the inputs: the table heading row text and the cell array of table data, and then formats the table.

```

import mlreportgen.report.*
import mlreportgen.dom.*

rpt = Report('testResults_cell','pdf');

testData_raw = {'Test 1',1.25,'Pass';'Test 2',1.43,...
    'Fail';'Test 3',1.51,'Pass';'Test 4',2.17,'Fail'};

% Obtain cell array size
[nrows,ncols] = size(testData_raw);

% Preallocate memory for cell array
testData{nrows,ncols} = [];

```

```

% Convert all values to strings to control number of
% decimal places displayed
testData = testData_raw;
idx = cellfun(@isnumeric, testData_raw(:));
testData(idx) = cellfun(@(x){sprintf('%.2f', x)}, testData_raw(idx));

% Set color of results column text items
for i = 1:nrows
    for j = 1:ncols
        d = string(testData(i,j));
        p = Paragraph(d);
        if d == "Pass"
            p.Color = 'green';
        elseif d == "Fail"
            p.Color = 'red';
        end
        testData(i,j) = {p};
    end
end

% Create and format table
table = FormalTable({'Name', 'Time', 'Result'}, testData);
table.RowSep = 'Solid';
table.ColSep = 'Solid';
table.Border = 'Solid';
table.TableEntriesInnerMargin = '5px';
table.Header.Style = {Bold()};
add(rpt, table);

close(rpt);
rptview(rpt);

```

## See Also

mlreportgen.report.Report | mlreportgen.dom.FormalTable |  
mlreportgen.dom.TableEntry | struct | cell

## More About

- “Create Informal Tables” on page 13-64

## Create and Format Images

### In this section...

“Create an Image” on page 13-88

“Resize an Image” on page 13-88

“Image Storage” on page 13-89

“Links from an Image” on page 13-89

### Create an Image

To create an image to a report, create an `mlreportgen.dom.Image` object. You can append it to one of these document element objects:

- Document
- Group
- Paragraph
- ListItem
- TableEntry

For example, you can create a MATLAB figure, save it as an image, and add the image to a report.

```
import mlreportgen.dom.*
d = Document('imageArea', 'html');

p = Paragraph('Plot 1');
p.Bold = true;
append(d,p);

x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y);

saveas(gcf, 'myPlot_img.png');

plot1 = Image('myPlot_img.png');
append(d,plot1);

close(d);
rptview(d.OutputPath);
```

For a list of supported image formats, see `mlreportgen.dom.Image`.

### Resize an Image

To resize an image object, you can:

- Set the `Image.Height` and `Image.Width` properties.
- Use an `mlreportgen.dom.Height` or `mlreportgen.dom.Width` object in an `Image.Style` property definition.

For Microsoft Word and PDF reports, you can use an `mlreportgen.dom.ScaleToFit` object to scale an image so that it fits within the page margins or in a table entry that contains it.

## Image Storage

Keep the original file until it has been copied into the document. The DOM API copies the contents of the source image file into the output document when you close the document.

## Links from an Image

You can specify an area in an image as a link. Clicking a link area in an image in an HTML browser opens the link. For details, see “Create Image Maps” on page 13-99.

## See Also

`mlreportgen.dom.Image` | `mlreportgen.dom.Height` | `mlreportgen.dom.Width` | `mlreportgen.dom.ScaleToFit`

## Related Examples

- “Add Content to a Report” on page 13-10

## More About

- “Report Formatting Approaches” on page 13-17

## Create a Title Page

You can add a title page to your report using the `mlreportgen.report.TitlePage` class. It is much easier and more efficient to use this class rather than using DOM objects to create and add a title page to your report. This class has predefined holes and formatting for the:

- Title
- Subtitle
- Image
- Author
- Publisher
- Publication date

You can exclude items you do not want on your title page and you can edit the `TitlePage` template to add more items.

This example shows code that creates a title page, which uses default formatting.

```
import mlreportgen.report.*
rpt = Report('output', 'pdf');

tp = TitlePage();
tp.Title = 'Aircraft Tests';
tp.Subtitle = 'Monthly Data';
tp.Image = which('b747.jpg');
tp.Author = 'John Smith';
tp.Publisher = 'MathWorks';
tp.PubDate = date();

add(rpt, tp);
close(rpt);
rptview(rpt);
```

# Aircraft Tests

## Monthly Data



**John Smith**

MathWorks

19-Jan-2018

For information and more examples, see `mlreportgen.report.TitlePage`.

## Create a Table of Contents

You can add a table of contents to your report using the `mlreportgen.report.TableOfContents` class. This predefined class automatically adds a formatted table of contents that contains the report headings into your report. It is much easier and more efficient to use this class rather than using DOM objects to create a table of contents. For information and examples, see `mlreportgen.report.TableOfContents`.

Alternately, using DOM objects, you can create a table of contents in your report program or you can use a template to define the TOC. To create the TOC programmatically, append an `mlreportgen.dom.TOC` object to your report document.

Using a template ensures that all report programs that use that template create the same type of TOC. Also, with a template, you update the TOC in only one place if your formatting changes.

If you are using a template, you can either:

- Include the TOC reference in your Word template or in your HTML or PDF template package (`root.html`).
- Create a document part template for the TOC and insert the document part programmatically.

Using either approach, your report program must create heading objects that specify a numeric level or paragraph objects that specify outline level. The TOC generator uses content with level information to define the structure.

## Create a TOC in Your Report Program

The DOM API supports automatic generation of a document's table of contents. To enable automatic TOC generation:

- Use `Paragraph` or heading objects (`Heading`, `Heading1`, and so on) in your document to specify section headings. If you use a `Paragraph` object for a heading, you must set the paragraph's `OutlineLevel` property to an appropriate value, for example, 1 for a chapter or other top-level heading.
- Insert a TOC placeholder in your document where you want to generate the TOC. You can insert a TOC placeholder programmatically or in the template for your document.

### Create a TOC Programmatically

To create a TOC placeholder programmatically, append an `mlreportgen.dom.TOC` object where you want to generate the TOC. You can specify the number of levels to include in the TOC and the type of leader. The default values are three levels and a dot leader. This example uses two levels and a dot leader.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'pdf');
open(d);

title = append(d, Paragraph('My TOC Document'));
title.Bold = true;
title.FontSize = '28pt';

toc = append(d, TOC(2));
```



```

toc.Style = {PageBreakBefore(true)};

h1 = append(d,Heading1('Chapter 1'));
h1.Style = {PageBreakBefore(true)};
p1 = append(d,Paragraph('Hello World'));

h2 = append(d,Heading2('Section 1.1'));
h2.Style = {PageBreakBefore(true)};
p2 = append(d,Paragraph('Another page'));

h3 = append(d,Heading3('My Subsection 1.1.a'));
p3 = append(d, Paragraph('My Level 3 Heading Text'));

close(d);
rptview(d.OutputPath);

```

## Use a Template to Create a Microsoft Word TOC

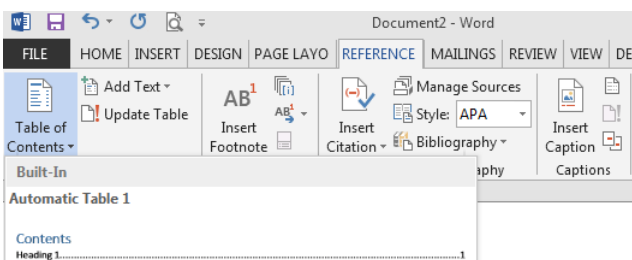
You can use Word to insert a Word TOC reference object in a Word document or document part template. Word replaces the TOC reference with an automatically generated TOC when it updates the document.

To generate a table of contents in a DOM Word report using a template containing a TOC reference:

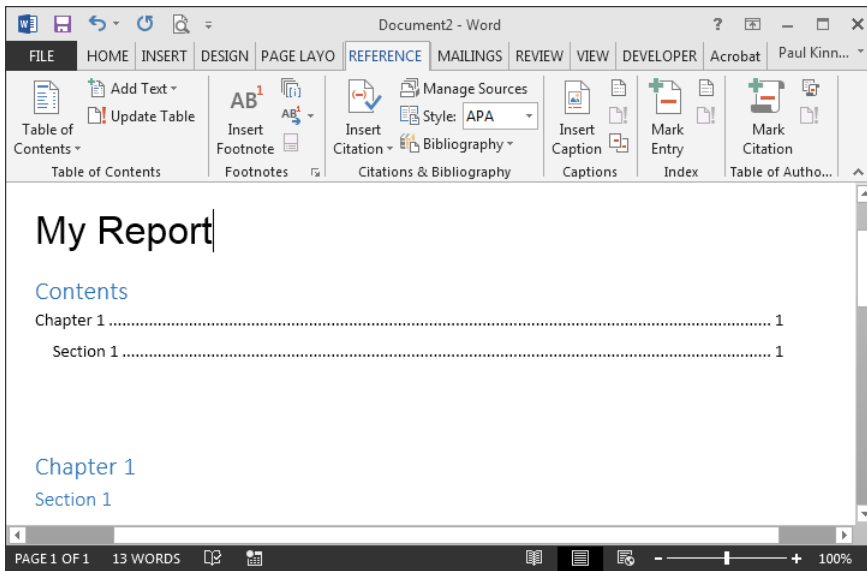
- 1 To specify where in the report to generate the TOC, create a table of contents reference in the Word template. See “Create a Word Table of Contents Reference” on page 13-93.
- 2 Set the outline levels of the section heads that you want to appear in the table of contents. See “Set Outline Levels of Section Heads” on page 13-96.
- 3 Update the generated document. See “Update the TOC in a Word Report” on page 13-94.

### Create a Word Table of Contents Reference

- 1 Open the template in Word.
- 2 Click where you want to create the table of contents.
- 3 On the **References** tab, click **Table of Contents**.



- 4 Select a TOC format option to generate a table of contents. For example, select the **Built-In** format option. The TOC appears.



- 5 Save the template.

---

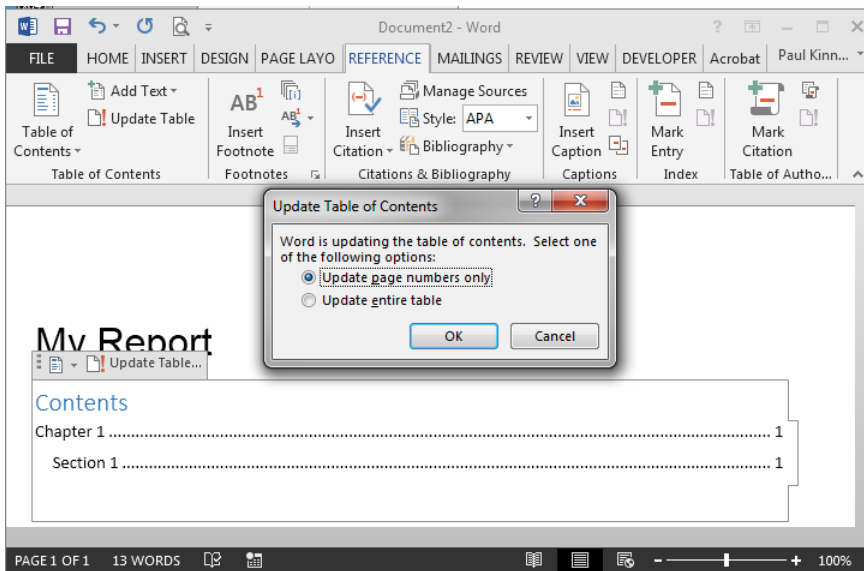
**Note** If you want to use a document part to insert a TOC, insert the TOC reference in the template for the document part. Delete the instance from the template before you save. See “Create a Microsoft Word Document Part Template Library” on page 13-27 and “Insert TOC Placeholder Programmatically Using a Document Part” on page 13-96.

---

### Update the TOC in a Word Report

You must update a Word document containing a TOC reference to generate a TOC. On Windows systems, the DOM API `rptview` command uses Word to update the Word document that it displays. If you open a Word document directly, for example, a document generated by the DOM API on system other than Windows, you must update the TOC.

- 1 In the Word template, select the TOC reference.
- 2 On the **References** tab, click **Update Table**.
- 3 In the Update Table of Contents dialog box, select **Update entire table** and click **OK**.



## Create Table of Contents in HTML or PDF Templates

When you use a PDF or HTML template to add a TOC, you can:

- Include a table of contents placeholder in the main template (root.html) in your template package.
- Include the TOC placeholder in a document part template.

To create a table of contents in an HTML or PDF report using a document part template:

- Define a document part template that includes the TOC placeholder.
- Programmatically insert the document part into your report.
- Use Paragraph or heading objects (Heading, Heading1, and so on) to specify your report's headings. If you use a Paragraph object for a heading, you must set its OutlineLevel property to an appropriate value.

### Define a Document Part Template for TOC

To create or modify a document part template for a TOC, use code in this form:

```
<dptemplate name="ReportTOC">
  <TOC number-of-levels ="3" leader-pattern="dots" />
</dptemplate>
```

You can:

- Replace ReportTOC with the name you prefer
- Set number-of-levels to the number of levels of headings you want to include in the TOC
- Set leader-pattern to dots or to spaces

For an example, see “PDF and HTML Document Parts and Holes” on page 13-134.

### Insert TOC Placeholder Programmatically Using a Document Part

Use the `DocumentPart` class to insert an instance of the document part that contains the TOC placeholder. If you define the document part template in your template, include the template package name when you define the document object. For example:

```
d = Document('MyReport', 'html', 'MyTemplate');
```

This code uses the supplied document part `ReportTOC` in the default template to generate a table of contents.

```
import mlreportgen.dom.*;
d = Document('MyReport', 'pdf');
append(d, 'My Report');

append(d, DocumentPart(d, 'ReportTOC'));
append(d, Heading1('Chapter 1'));
append(d, Heading2('Section 1'));

close(d);
rptview(d.OutputPath);
```

---

**Tip** Use the variable assigned to the `Document` object in the `DocumentPart` syntax to use the document part template associated with the document object:

```
append(d, DocumentPart(d, 'ReportTOC'));
```

If you use this syntax, you do not need to call the template and specify the document type when you refer to the document part. This approach simplifies your code and generates the report more efficiently.

---

### Set Outline Levels of Section Heads

To generate a table of contents in your report, your program must set the outline levels of the section heads that you want in the contents. An outline level is a paragraph format property that specifies whether and at what level a paragraph's contents appear in a table of contents. For example, if a paragraph has an outline level of 1, the content appears at the top level of the generated table of contents. You can specify up to nine outline levels.

To set the outline level of paragraphs, use one of these approaches.

- “Use Template-Defined Styles to Set Outline Levels” on page 13-96
- “Use Format Objects to Set Outline Levels” on page 13-97
- “Use Heading Objects to Set Outline Levels” on page 13-97

#### Use Template-Defined Styles to Set Outline Levels

You can use styles defined in the report's template to set the outline level of a paragraph. By default Word documents include a set of styles, `Heading 1`, `Heading 2`, and so on, that define outline levels. Your program can use these built-in styles to specify for these heads to appear in the TOC. This example uses template-defined styles to set the outline levels of section heads and assumes that the template `MyTemplate` includes a TOC reference.

```
import mlreportgen.dom.*;
d = Document('MyReport', 'docx', 'MyTemplate');

append(d, Paragraph('Chapter 1', 'Heading 1'));
append(d, Paragraph('Section 1', 'Heading 2'));

close(d);
rptview(d.OutputPath); % Updates the TOC
```

You can also use Word or an HTML editor to define your own heading styles and then use them to generate a report.

### Use Format Objects to Set Outline Levels

You can use format objects to set outline levels. This example assumes that the template `MyTemplate` includes a TOC reference.

```
import mlreportgen.dom.*;
d = Document('MyReport', 'docx', 'MyTemplate');

h1 = {FontFamily('Arial'),FontSize('16pt'),OutlineLevel(1)};
h2 = {FontFamily('Arial'),FontSize('14pt'),OutlineLevel(2)};
p = append(d, Paragraph('Chapter 1'));
p.Style = h1;
p = append(d, Paragraph('Section 1'));
p.Style = h2;

close(d);
rptview(d.OutputPath); % Updates the TOC
```

### Use Heading Objects to Set Outline Levels

You can use `mlreportgen.dom.Heading1` object (and `Heading2`, `Heading3`, and so on) to specify outline levels. A `Heading1` object is a paragraph whose constructor specifies its outline level. You can use a `Heading1` object alone or with template-based styles or format-object-based styles. This example assumes that the template `MyTemplate` includes a TOC reference.

```
import mlreportgen.dom.*;
d = Document('MyReport', 'docx', 'MyTemplate');

h1 = {FontFamily('Arial'),FontSize('16pt')};
h2 = {FontFamily('Arial'),FontSize('14pt')};
h = append(d, Heading1('Chapter 1'));
h.Style = h1;
h = append(d, Heading2('Section 1'));
p.Style = h2;

close(d);
rptview(d.OutputPath); % Updates the TOC
```

In HTML and PDF reports, the `Heading1` and `Heading2` objects generate the HTML elements `h1` and `h2`. By using the objects `Heading1`, `Heading2`, and so on, you can ensure that your report uses the default styles for headings.

## **See Also**

### **Functions**

zipTemplate | unzipTemplate | rptview

### **Classes**

mlreportgen.dom.Heading | mlreportgen.dom.Heading1

## **Related Examples**

- “Create a Microsoft Word Template” on page 13-119
- “Create an HTML or PDF Template” on page 13-130

## Create Image Maps

In an HTML or PDF report, you can specify areas of an image as links. Clicking the link area in an image in an HTML browser opens the target. You can map different areas in an image to different link targets.

- 1 Create an `mlreportgen.dom.ImageArea` object for each image area that is to serve as a link. You can specify text to display if the image is not visible.

You can specify an image area to have one of these shapes:

- Rectangle
- Circle
- Polygon

For details, see `mlreportgen.dom.ImageArea`.

- 2 Create an `mlreportgen.dom.ImageMap` object to associate the link areas with the image. Append the `ImageArea` objects to the `ImageMap` object.

For example, you can create a link from a plot image to documentation about plotting.

```
import mlreportgen.dom.*
d = Document('imageArea', 'pdf');
open(d);

% Set page size to A4
pageSize = d.CurrentPageLayout.PageSize;
pageSize.Height = '297mm';
pageSize.Width = '230mm';

% Set margins to 0
pageMargins = d.CurrentPageLayout.PageMargins;
pageMargins.Top = '0mm';
pageMargins.Bottom = '0mm';
pageMargins.Left = '0mm';
pageMargins.Right = '0mm';

% Create a plot and save it as an image file
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y);
annotation('textbox', [0.2,0.4,0.1,0.1],...
    'string', 'Help on plot function');
saveas(gcf, 'plot_img.png');

% Create the DOM image object and append it to your document
plot1 = Image('plot_img.png');
append(d, plot1);

% Define the area and link target using ImageArea
target = ['https://www.mathworks.com/help/matlab/ref/' ...
    'plot.html?searchHighlight=plot'];
area1 = ImageArea( target, ...
    'plot function help', 160, 340, 383, 392);
```

```
% Create the image map object and append the area to it
map = ImageMap();
append(map,areal);
plot1.Map = map;

close(d);
rptview(d.OutputPath);
```

### See Also

#### Classes

`mlreportgen.dom.ImageMap` | `mlreportgen.dom.ImageArea` | `mlreportgen.dom.Image`

#### Functions

### Related Examples

- “Add Content to a Report” on page 13-10

### More About

- “Report Formatting Approaches” on page 13-17



# Automatically Number Document Content

## In this section...

“Automatically Number Content Programmatically” on page 13-101

“Automatically Number Content Using Part Templates” on page 13-102

You can automatically number document content, such as chapter, section, table, and figure headings. Append automatic numbering objects to the document where you want numbers to appear. Each automatic number is associated with a numbering stream that determines the value of each number in a sequence. Report generation replaces an automatic numbering object with a number based on its position in the document relative to other automatic numbers in the same stream. For example, the first automatic numbering object in a stream is replaced with 1, the second with 2, and so on. You can use automatic numbering to create hierarchical numbering schemes such as Section 1.1 and Section 1.2.

You can automatically number document content programmatically or by defining the autonumbers in a template.

## Automatically Number Content Programmatically

To automatically number document content programmatically, do the following at each point in a document where you want an automatically generated number to appear.

- 1 Create an automatic numbering object, using the `m1reportgen.dom.AutoNumber` constructor. Specify the name of the associated automatic numbering stream in the constructor. For example, this line creates an automatic number belonging to the stream named `chapter`.

```
chapterNumber = AutoNumber('chapter');
```

---

**Note** If the specified automatic numbering stream does not exist, the `AutoNumber` constructor creates a numbering stream having the specified name. The implicitly constructed stream renders automatic numbers as Arabic numerals. To use a stream with different properties, create the stream explicitly, using a `createAutoNumberStream` function of a `Document` object.

---

- 2 Append the `AutoNumber` to a `Text`, `Paragraph`, or `Heading` object that contains the text that precedes the automatic number.
 

```
append(chapHead, chapterNumber);
```
- 3 Append an `m1reportgen.dom.CounterInc` format object to the `Style` property of the content object that you want to automatically number. Appending a `CounterInc` object increments the stream associated with the automatic number when the paragraph or heading is output. The updated value replaces the `AutoNumber` object.

```
chapHead.Style = {CounterInc('chapter'), WhiteSpace('preserve')};
```

This code automatically numbers the chapter headings in a document.

```
import m1reportgen.dom.*;
d = Document('MyReport', 'html');

for rank = 3:5
    chapHead = Heading1('Chapter ', 'Heading 1');
```

```

    append(chapHead,AutoNumber('chapter'));
    append(chapHead,sprintf('. Rank %i Magic Square',rank));
    chapHead.Style = {CounterInc('chapter'), ...
        WhiteSpace('preserve')};
    append(d,chapHead);
    table = append(d,magic(rank));
    table.Width = '2in';
end

close(d);
rptview(d.OutputPath);

```

### Create Hierarchical Automatic Numbering

You can create hierarchical numbering schemes, such as 1.1, 1.2, 1.3, 2.1, and 2.2. Use an `mlreportgen.dom.CounterReset` format object to reset a child automatic number to its initial value when its parent number changes. For example, this code uses a `CounterReset` format object to reset the chapter table number stream at the beginning of each chapter.

```

import mlreportgen.dom.*;
d = Document('MyReport','html');

for rank = 3:2:9
    chapHead = Heading(1,'Chapter ');
    append(chapHead, AutoNumber('chapter'));
    chapHead.Style = {CounterInc('chapter'), ...
        CounterReset('table'), ...
        WhiteSpace('preserve')};
    append(d,chapHead);

    for i = 0:1;
        tableHead = Paragraph('Table ');
        append(tableHead,AutoNumber('chapter'))
        append(tableHead, '. ');
        append(tableHead, AutoNumber('table'));
        append(tableHead, ...
            sprintf('. Rank %i Magic Square',rank+i));
        tableHead.Style = {CounterInc('table'), ...
            Bold, ...
            FontSize('11pt'), ...
            WhiteSpace('preserve')};
        append(d,tableHead);
        table = append(d,magic(rank+i));
        table.Width = '2in';
    end
end

close(d);
rptview(d.OutputPath);

```

### Automatically Number Content Using Part Templates

You can automatically number a document by creating a document part object based on templates containing Microsoft Word, HTML, or PDF automatic numbering and repeatedly appending the parts to a document.

## Automatic Numbering in Word Reports

Suppose that you add a chapter part template `Chapter` to the part template library of the Word `MyReportTemplate.dotx` report template. This template uses a Word sequence (SEQ) field to number the chapter heading. The template also contains holes for the chapter title and the chapter content.

```
Chapter { SEQ Chapter \* MERGEFORMAT } ChapterTitle ChapterContent
ChapterContent ChapterContent ChapterContent
```

This code uses the chapter part template to create numbered chapters. The last statement in this code opens the report in Word and updates it. Updating the report causes Word to replace the SEQ fields with the chapter numbers.

```
import mlreportgen.dom.*
doctype = 'docx';
d = Document('MyReport',doctype,'MyReportTemplate');

for rank = 3:5
    chapterPart = DocumentPart(d,'Chapter');
    while ~strcmp(chapterPart.CurrentHoleId,'#end#')
        switch chapterPart.CurrentHoleId
            case 'ChapterTitle'
                append(chapterPart, ...
                    sprintf('Rank %i Magic Square',rank));
            case 'ChapterContent'
                table = append(chapterPart,magic(rank));
                table.Width = '2in';
        end
        moveToNextHole(chapterPart);
    end
    append(d, chapterPart);
end

close(d);
rptview(d.OutputPath);
```

## Automatic Numbering in HTML Reports

To create automatic numbering in HTML reports, create a document part that uses the counter-increment property, and define the counter in the style sheet. For example, to create a document part to work with the same program used in “Automatic Numbering in Word Reports” on page 13-103, create a document part template in your HTML document library similar to this code. The code defines the chapter counter and specifies a class `an_chapter` to hold the autonumber. It also defines holes for the title and for the content to work with the program.

```
<dptemplate name="Chapter">
  <p style="counter-increment:chapter;"><span>Chapter </span>
    <span class="an_chapter"></span>
  <hole id="ChapterTitle" /></p>
  <hole id="ChapterContent" />
</dptemplate>
```

In the style sheet, define the `an_chapter` class. Use the content property to specify the chapter counter as the content.

```
span.an_chapter:before {  
  content: counter(chapter);  
}
```

Use the same program as you used for Word, changing the value for `doctype` to `'html'`.

### Automatic Number in PDF Reports

Creating automatic numbers for PDF is similar to HTML, except the DOM API provides an HTML element `<autonumber>` for PDF templates that simplifies automatic numbering. Specify the `stream-name` attribute for the `autonumber` element. For the stream name, use the value of a `counter-increment` property, in this case `chapter`.

```
<dptemplate name="Chapter">  
  <p style="counter-increment:chapter;"><span>Chapter </span>  
    <autonumber stream-name="chapter"/>  
  <hole id="ChapterTitle" /></p>  
  <hole id="ChapterContent" />  
</dptemplate>
```

You do not need to add properties in the style sheet to use the `autonumber`.

Use the same program you used for Word, changing the value for `doctype` to `'pdf'`.

## See Also

### Functions

`createAutoNumberStream`

### Classes

`mlreportgen.dom.AutoNumberStream` | `mlreportgen.dom.AutoNumber` |  
`mlreportgen.dom.CounterInc` | `mlreportgen.dom.CounterReset`

## Convert HTML Content to a DOM Object

You can convert HTML content to a DOM object that you can add to a report. The HTML content can be in a string or a file. To convert HTML content that is in a string, use one of these approaches:

- Create an `mlreportgen.dom.HTML` object from the string of HTML content and add the object to a report. You can use this approach with a DOM API or Report API report.
- Convert the HTML content and add it to an `mlreportgen.dom.Document` or `mlreportgen.dom.DocumentPart` object by using the `addHTML` method. This method returns an HTML object. You can use this approach only with a DOM API report.

See “Convert HTML Content in a String” on page 13-106.

To convert HTML content that is in a file, use one of these approaches:

- Create an `mlreportgen.dom.HTMLFile` object from the HTML file and add the object to a report. You can use this approach with a DOM API or Report API report.
- Convert the HTML content and add it to an `mlreportgen.dom.Document` or `mlreportgen.dom.DocumentPart` object by using the `addHTMLFile` method. This method returns an `HTMLFile` object. You can use this approach only with a DOM API report.

See “Convert HTML File Content” on page 13-106.

### Prepare HTML Before Conversion

MATLAB Report Generator `mlreportgen.dom.HTML` and `mlreportgen.dom.HTMLFile` objects typically cannot accept the raw HTML output of third-party applications, such as Microsoft Word, that export native documents as HTML markup. In these cases, your Report API report generation program can use the `mlreportgen.utils.html2dom.prepHTMLString` and `mlreportgen.utils.html2dom.prepHTMLFile` functions to prepare the raw HTML for use with the `mlreportgen.dom.HTML` or `mlreportgen.dom.HTMLFile` objects. These functions:

- Correct invalid markup by calling `mlreportgen.utils.tidy` with the settings for HTML output.
- Use the MATLAB web browser to convert the tidied markup to an HTML DOM document. See <https://www.w3.org/TR/WD-DOM/introduction.html>.

The MATLAB web browser computes the CSS properties of the elements in the HTML input based on internal and external style sheets specified by the input HTML, and on the style attribute of an element. The CSS property computation supports all valid CSS style sheet selectors, including selectors not directly supported by `mlreportgen.dom.HTML` and `mlreportgen.dom.HTMLFile` objects.

- Converts the HTML DOM document to HTML markup that is supported by `mlreportgen.dom.HTML` and `mlreportgen.dom.HTMLFile` objects. The style attribute for each element specifies the element CSS properties that the MATLAB web browser computed.

Typically, your program will have to further process the prepared HTML to remove valid but undesirable objects, such as line feeds that were in the raw content.

For an example that prepares HTML content from a file, see “Prepare HTML for Conversion to a DOM Object” on page 13-108.

## Convert HTML Content in a String

To convert HTML content in a string to a DOM object, create an `mlreportgen.dom.HTML` object and add the object to the report.

```
import mlreportgen.dom.*;
d = Document('MyDoc', 'docx');
htmlObj = HTML('<p><b>Hello </b> <i style="color:green">World</i></p>');
append(d,htmlObj);
close(d);
rptview(d);
```

Alternatively, convert the HTML and add it to the document by using the `addHTML` method. The method returns an HTML object.

```
import mlreportgen.dom.*;
d = Document('MyDoc', 'docx');
addHTML(d, '<p><b>Hello </b> <i style="color:green">World</i></p>');
close(d);
rptview(d);
```

Once you create an HTML object, you can append more HTML content to the object. For example:

```
import mlreportgen.dom.*;
d = Document('MyDoc', 'docx');
htmlObj = HTML('<p><b>Hello </b> <i style="color:green">World</i></p>');

append(htmlObj, '<p>This is <u>me</u> speaking</p>');
append(d,htmlObj);

close(d);
rptview(d);
```

To append the content of an HTML object more than once in a report, use the `clone` method with the HTML object. Then, append the clone to the report.

## Convert HTML File Content

To convert HTML file content to a DOM object, create an `mlreportgen.dom.HTMLFile` object and add the object to the report.

Create a file, `MyHTML.html`, that contains this HTML:

```
<html><p style="color:green;font-family:arial">Hello World</p></html>
```

Generate a PDF report based on the contents of the HTML file.

```
import mlreportgen.dom.*;
d = Document('MyPDF', 'pdf');
htmlObj = HTMLFile('MyHTML.html');
append(d,htmlObj);
close(d);
rptview(d);
```

Alternatively, convert the HTML and add it to the document by using the `addHTMLFile` method.

```
import mlreportgen.dom.*;
d = Document('MyPDF', 'pdf');
```

```
addHTMLFile(d, 'MyHTML.html');  
close(d);  
rptview(d);
```

## See Also

[addHTML](#) | [addHTMLFile](#) | [mlreportgen.dom.HTMLFile](#) | [mlreportgen.dom.HTML](#) | [mlreportgen.utils.html2dom.prepHTMLFile](#) | [mlreportgen.utils.html2dom.prepHTMLString](#) | [mlreportgen.utils.tidy](#)

## More About

- “Requirements for Converting HTML to a DOM Object” on page 13-110
- “Prepare HTML for Conversion to a DOM Object” on page 13-108

## Prepare HTML for Conversion to a DOM Object

This example shows how to prepare HTML content before converting it to the MATLAB Report Generator internal document object model (DOM). The example creates a CSS style sheet and an HTML file that uses the style sheet. The example uses the `mlreportgen.utils.html2dom.prepHTMLFile` function to prepare the HTML file content so that it can be converted to an `mlreportgen.dom.HTMLFile` object. The function tidies the HTML and converts the HTML to markup that is supported by an `mlreportgen.dom.HTMLFile` object. See “Prepare HTML Before Conversion” on page 13-105. To prepare HTML content that is in a string, use `mlreportgen.utils.html2dom.prepHTMLString`.

Create a CSS style sheet, `myCSS.css`, to specify that the text in a paragraph is red.

```
p {
    color: red;
}
```

Create a file, `myHTML.html`, that contains this HTML:

```
<html>
<head>
  <link rel="stylesheet" type="text/css" href="myCSS.css" >
</head>
<body>
  <p> Hello World</p>
</body>
</html>
```

The HTML cannot be used by a MATLAB Report Generator DOM `HTMLFile` object because the `link` element is not properly closed. The slash (/) before the closing angle bracket (>) is missing.

Try to convert the HTML to an `mlreportgen.dom.HTMLFile` object.

```
import mlreportgen.dom.*
d = Document('test','pdf');
htmlObj = HTMLFile('myHTML.html');
append(d,htmlObj);
close(d);
rptview(d);
```

```
Error using mlreportgen.dom.HTMLFile
HTML error: expected end of tag 'link'
```

Tidy the HTML by using `mlreportgen.utils.tidy`. This function does not format the HTML elements using the formatting in the style sheet.

```
import mlreportgen.dom.*
import mlreportgen.utils.*
d = Document('test','pdf');
tidy('myHTML.html');
htmlObj = HTMLFile('myHTML-tidied.html');
append(d,htmlObj);
close(d);
rptview(d);
```

In the generated report, the text is black, not red.



## Hello World

Use `mlreportgen.utils.html2dom.prepHTMLFile` to tidy the HTML and format the HTML elements according to the style sheet.

```
import mlreportgen.dom.*
import mlreportgen.utils.html2dom.*
d = Document('test','pdf');
preppedHTMLFile = prepHTMLFile('myHTML.html','mypreppedHTML.html');
htmlObj = HTMLFile(preppedHTMLFile);
append(d,htmlObj);
close(d);
rptview(d);
```

In the generated report, the text is red.

## Hello World

---

**Note** In the process of preparing HTML, the `mlreportgen.utils.html2dom.prepHTMLFile` and `mlreportgen.utils.html2dom.prepHTMLString` functions load the HTML in a MATLAB web browser. See “Prepare HTML Before Conversion” on page 13-105. Formatting that is not specified in the input HTML or style sheet is determined by the browser. The input HTML in this example does not specify the font family. Therefore, the font family is determined by the browser. To render the text with a different font, you can specify the font in the input style sheet or HTML before preparing the HTML conversion or use the report generation formatting capabilities. See “Report Formatting Approaches” on page 13-17.

---

## See Also

`mlreportgen.utils.html2dom.prepHTMLFile` |  
`mlreportgen.utils.html2dom.prepHTMLString` | `mlreportgen.utils.tidy` |  
`mlreportgen.dom.HTMLFile` | `mlreportgen.dom.HTML`

## More About

- “Requirements for Converting HTML to a DOM Object” on page 13-110
- “Convert HTML Content to a DOM Object” on page 13-105

## Requirements for Converting HTML to a DOM Object

To convert HTML content to an `mlreportgen.dom.HTML` or `mlreportgen.dom.HTMLFile` object, the HTML content must be XML parsable. HTML content is XML parsable when it complies with the rules for properly formed XML, such as:

- Include a closing tag for all elements.
- Use lower case for the opening and closing (start and end) tags of an element. For example, use `<p>` and `</p>` for a paragraph element, not `<P>` and `</P>`.
- Nest elements properly. If you open an element inside another element, close the first element before you close the containing element.
- Enclose attribute values with quotation marks. For example, use `<p align="center"></p>`.

For details, see the W3Schools summary of XML rules at [www.w3schools.com/xml/xml\\_syntax.asp](http://www.w3schools.com/xml/xml_syntax.asp).

---

**Tip** To make HTML content XML parsable, you can use `mlreportgen.utils.html2dom.prepHTMLString`, `mlreportgen.utils.html2dom.prepHTMLFile`, and `mlreportgen.utils.tidy`. See “Prepare HTML Before Conversion” on page 13-105.

---

## Supported HTML Elements and Attributes

This table shows the HTML elements and attributes that are supported when you convert HTML to a DOM object. Unsupported elements and attributes are ignored.

HTML Element	Attributes
a	class, style, href, name
address	class, style
b	class, style
big	class, style
blockquote	class, style
body	class, style
br	n/a
center	class, style
cite	class, style
code	class, style
dd	class, style
del	class, style
dfn	class, style
div	class, style
dl	class, style
dt	class, style
em	class, style

HTML Element	Attributes
font	class, style, color, face, size
h1, h2, h3, h4, h5, h6	class, style, align
hr	class, style, align
i	class, style
ins	class, style
img	class, style, src, height, width
kbd	class, style
li	class, style
mark	class, style
nobr	class, style
ol	class, style
p	class, style, align
pre	class, style
s	class, style
samp	class, style
small	class, style
span	class, style
strike	class, style
strong	class, style
sub	class, style
sup	class, style
table	class, style, align, bgcolor, border, cellpadding, cellspacing, frame, rules, width
tbody	class, style, align, valign
tfoot	class, style, align, valign
thead	class, style, align, valign
td	class, style, bgcolor, height, width, colspan, rowspan, align, valign, nowrap
th	class, style, bgcolor, height, width, colspan, rowspan, align, valign, nowrap
tr	class, style, align, bgcolor, valign
tt	class, style
u	class, style
ul	class, style
var	class, style

For information about these elements, see <https://developer.mozilla.org/en-US/docs/Web/HTML/Element>.

## Supported HTML CSS Style Attributes for All Elements

You can use HTML style attributes to format HTML content that you append to a DOM report. A style attribute is a string of Cascading style sheets (CSS) formats.

These CSS formats are supported:

- background-color
- border
- border-bottom
- border-bottom-color
- border-bottom-style
- border-bottom-width
- border-color
- border-left
- border-left-color
- border-left-style
- border-left-width
- border-right
- border-right-color
- border-right-style
- border-right-width
- border-style
- border-top
- border-top-color
- border-top-style
- border-top-width
- border-width
- color
- counter-increment
- counter-reset
- display
- font-family
- font-size
- font-style
- font-weight
- height
- line-height
- list-style-type
- margin
- margin-bottom

- `margin-left`
- `margin-right`
- `margin-top`
- `padding`
- `padding-bottom`
- `padding-left`
- `padding-right`
- `padding-top`
- `text-align`
- `text-decoration`
- `text-indent`
- `vertical-align`
- `white-space`
- `width`

For information about these formats, <https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>.

## Support for HTML Character Entities

You can append HTML content that includes special characters, such as the British pound sign, the U.S. dollar sign, or reserved XML markup characters. The XML markup special characters are `>`, `<`, `&`, `"`, and `'`. To include special characters, use HTML named or numeric character references. For example, to include the left angle bracket (`<`) in HTML content that you want to append, use one of these character entity references:

- The named character entity reference `&lt;`;
- The numeric character entity reference `&003c;`

For more information, see [https://en.wikipedia.org/wiki/List\\_of\\_XML\\_and\\_HTML\\_character\\_entity\\_references](https://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references).

## DOCTYPE Declaration

The HTML content that you append to a DOM report does not need to include a document type declaration (see [https://en.wikipedia.org/wiki/Document\\_type\\_declaration](https://en.wikipedia.org/wiki/Document_type_declaration)). If the content includes a document type declaration, it must meet the following conditions:

- If the content includes character entity references (special characters), the document type declaration must reference a document type definition (DTD) that defines the referenced entities. For example, the following declaration specifies a DTD file that defines all HTML character entities:

```
<!DOCTYPE html SYSTEM "html.dtd">
```

The `html.dtd` is included in the MATLAB Report Generator software.

- If the document type declaration references a DTD file, a valid DTD file must exist at the path specified by the declaration. Otherwise, appending the content causes a DTD parse error. For example, the following declaration causes a parse error:

```
<!DOCTYPE html SYSTEM "foo.dtd">
```

- If the content to be appended does not include character entity references, the document type declaration does not need to reference a DTD file. For example, the following declaration works for content that does not use special characters:

```
<!DOCTYPE html>
```

---

**Tip** To avoid document type declaration issues, remove declarations from existing HTML content that you intend to append to DOM reports. If the content does not include a declaration, the DOM prepends a valid declaration that defines the entire HTML character entity set.

---

### See Also

### Related Examples

- “Prepare HTML for Conversion to a DOM Object” on page 13-108
- “Convert HTML Content to a DOM Object” on page 13-105

## Display Progress and Debugger Messages

### In this section...

- “Report Generation Messages” on page 13-115
- “Display DOM Default Messages” on page 13-115
- “Create and Display a Progress Message” on page 13-116

### Report Generation Messages

The DOM API includes a set of messages that can display when you generate a report. The messages are triggered every time a document element is created or appended during report generation.

You can define additional messages to display during report generation. The DOM API provides these classes for defining messages:

- `ProgressMessage`
- `DebugMessage`
- `WarningMessage`
- `ErrorMessage`

The DOM API provides additional classes for handling report message dispatching and display. It uses MATLAB events and listeners to dispatch messages. A message is dispatched based on event data for a specified DOM object. For an introduction to events and listeners, see “Event and Listener Concepts”.

---

**Note** When you create a message dispatcher, the DOM API keeps the dispatcher until the end of the current MATLAB session. Delete message event listeners to avoid duplicate reporting of message objects during a MATLAB session.

---

### Display DOM Default Messages

This example shows how to display the default DOM debug messages. Use a similar approach for displaying other kinds of DOM report messages.

- 1 Create a message dispatcher, using the `MessageDispatcher.getTheDispatcher` method. Use the same dispatcher for all messages.

```
dispatcher = MessageDispatcher.getTheDispatcher;
```

- 2 Use the `MessageDispatcher.Filter` property to specify to display debug messages.

```
dispatcher.Filter.DebugMessagesPass = true;
```

- 3 Add a listener using the MATLAB `addlistener` function. Specify the dispatcher object, the source and event data, and a `disp` function that specifies the event data and format to use for the message.

```
l = addlistener(dispatcher, 'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsText));
```

- 4 Include a code to delete the listener. Place it after the code that generates the report.

```
delete(l);
```

This report displays debug messages.

```
import mlreportgen.dom.*;
d = Document('test','html');

dispatcher = MessageDispatcher.getTheDispatcher;
dispatcher.Filter.DebugMessagesPass = true;

l = addlistener(dispatcher,'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsText));

open(d);

p = Paragraph('Chapter ');
p.Tag = 'chapter title';
p.Style = { CounterInc('chapter'),...
    CounterReset('table'),WhiteSpace('pre') };
append(p, AutoNumber('chapter'));
append(d,p);

close(d);
rptview('test','html');

delete(l);
```

## Create and Display a Progress Message

This example shows how to create and dispatch a progress message. You can use a similar approach for other kinds of messages, such as warnings.

- 1 Create a message dispatcher.

```
dispatcher = MessageDispatcher.getTheDispatcher;
```

- 2 Add a listener using the MATLAB `addlistener` function.

```
l = addlistener(dispatcher,'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsText));
```

- 3 Dispatch the message, using the `Message.dispatch` method. Specify the dispatcher object and the message to dispatch. Here the message is a debug message called `starting chapter`, and the Document object `d` is the source of the message.

```
dispatch(dispatcher,ProgressMessage('starting chapter',d));
```

- 4 Include code to delete the listener, after the code that generates the report.

```
delete(l);
```

This report uses this progress message.

```
import mlreportgen.dom.*;
d = Document('test','html');

dispatcher = MessageDispatcher.getTheDispatcher;

l = addlistener(dispatcher,'Message', ...
```



```
        @(src, evtdata) disp(evtdata.Message.formatAsText));

open(d);
dispatch(dispatcher,ProgressMessage('starting chapter',d));

p = Paragraph('Chapter ');
p.Tag = 'chapter title';
p.Style = { CounterInc('chapter'),...
    CounterReset('table'),WhiteSpace('pre') };
append(p, AutoNumber('chapter'));
append(d,p);

close(d);
rptview('test','html');

delete(l);
```

The MATLAB Command Window displays progress messages, including the starting chapter message and the messages the DOM API dispatches by default.

## See Also

### Functions

`dispatch` | `mlreportgen.dom.MessageDispatcher.getTheDispatcher` | `formatAsHTML` | `formatAsText` | `passesFilter`

### Classes

`mlreportgen.dom.MessageDispatcher` | `mlreportgen.dom.MessageEventData` | `mlreportgen.dom.MessageFilter` | `mlreportgen.dom.DebugMessage` | `mlreportgen.dom.ProgressMessage` | `mlreportgen.dom.WarningMessage` | `mlreportgen.dom.ErrorMessage`

## Create a Standalone Application from a Report Program

If you have the MATLAB Compiler™ product, you can package your DOM and Report API generation programs into standalone applications. Then you can share the standalone applications with others who do not have a licensed copy of MATLAB.

To enable execution of a standalone report generation program on a system that does not have a licensed copy of MATLAB, you must include `makeDOMCompilable()` before the first line of DOM or Report API code. For example:

```
makeDOMCompilable();  
  
import mlreportgen.dom.*  
  
d = Document("output","pdf");  
append(d,Text("Hello World"));  
close(d);
```

---

**Note** To enable execution of a standalone MATLAB API for PowerPoint (PPT API) program, use `makePPTCompilable()` instead of `makeDOMCompilable()`.

---

To create a standalone application from a report generation program, use the `compiler.build.standaloneApplication` function or the **Application Compiler** app. See “Create Standalone Application from MATLAB” (MATLAB Compiler).

### See Also

`compiler.build.standaloneApplication`

### More About

- “Create Standalone Application from MATLAB” (MATLAB Compiler)
- “Create a Standalone Application from a Presentation Program” on page 14-17

## Create a Microsoft Word Template

Use one of these approaches to create a Word template for generating a report.

- Use `mlreportgen.dom.Document.createTemplate` to create a copy of the DOM API default Word template that you can then customize. For example,

```
mlreportgen.dom.Document.createTemplate('mytemplate', 'docx');
```

- Use an existing Word template (for example, a report template for your organization) and customize the template to use with the DOM API.
- Create a Word template.

---

**Note** Word for Mac does not support creating holes for DOM API templates. If you need to create a Word template for generating Word documents on a Mac, you can:

- Create a template programmatically on the Mac, using the DOM API. See `mlreportgen.dom.Template` and `mlreportgen.dom.TemplateHole`.
  - Use Word on Windows to create your template and copy the template to your Mac.
- 

If you copy an existing template that is not based on the DOM API default Word template, apply any standard Word styles such as Title, Heading 1, TOC 1, List 1, and Emphasis to an element in the template. You can apply the styles to placeholder content and then remove the content. That process creates instances of the standard styles in the template style sheet.

For information about how to create templates and to copy styles from one template to another, see <https://support.microsoft.com/en-us/word>.

### See Also

#### Related Examples

- “Add Holes in a Microsoft Word Template” on page 13-120
- “Modify Styles in a Microsoft Word Template” on page 13-126
- “Create an HTML or PDF Template” on page 13-130

## Add Holes in a Microsoft Word Template

Holes are placeholders in a template that are filled with content as you generate a report. Use holes for dynamic content in a report. You can create two types of holes in a Microsoft Word report:

- **Inline hole** — A placeholder for content that a paragraph can contain, such as text, an image, or a link.
- **Block hole** — A placeholder for content that a paragraph cannot contain, such as a table or a list. A block hole can also contain content that an inline hole can contain, but in a Word report, the content is wrapped in a paragraph.

To add holes to a Word template, use rich text content controls. Before you add holes, make paragraph and formatting symbols visible and display the **Developer** tab.

### Open Word Template

Open the Word template file by using one of these methods:


- In MATLAB, in the **Current Folder** pane, right-click the template file and click **Open Outside MATLAB**.
- Outside of MATLAB, right-click the template file and click **Open**.

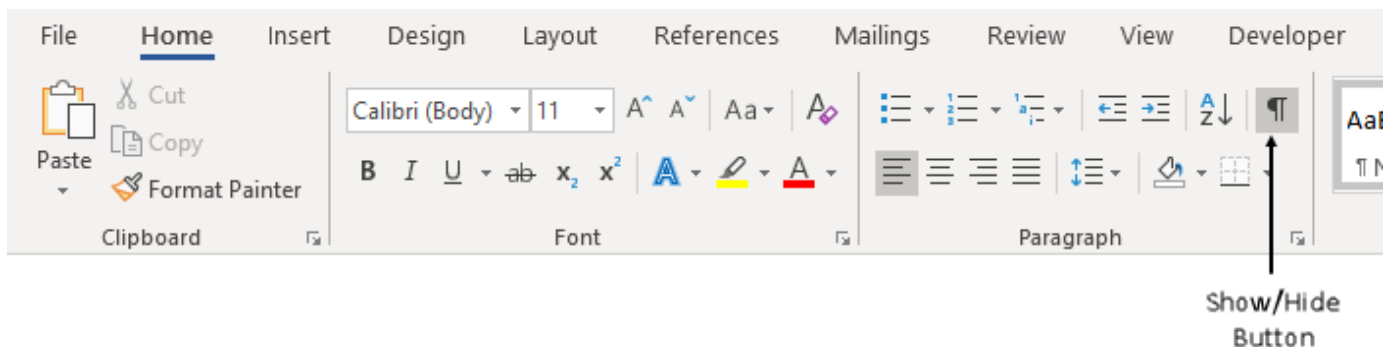
---

**Note** Do not double-click a Word template file to open it. Double-clicking the file opens a Word document file that uses the template.

---

### Display Formatting Symbols

To make paragraph and formatting symbols visible in Word, on the **Home** tab, in the **Paragraph** group, click the **Show/Hide** button .



The formatting symbols help you to see whether a hole is an inline hole or a block hole.

### Display the Developer Tab

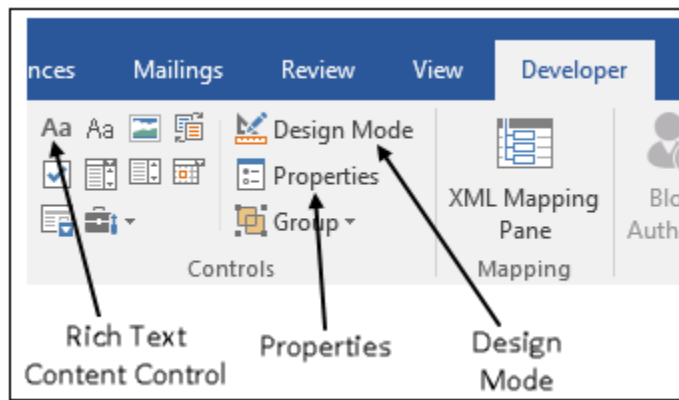
Before you can add a hole to a Word template, you must display the **Developer** tab.

If the **Developer** tab is not available, click **File > Options**, and then click **Customize Ribbon**. Under **Main Tabs**, click the **Developer** check box.

**Tip** If you do not see the **Developer** check box in the list, set **Customize the Ribbon** to **Main Tabs**.

When you create a hole, you use these controls on the **Developer** tab:

- Design Mode
- Rich Text Content Control
- Properties



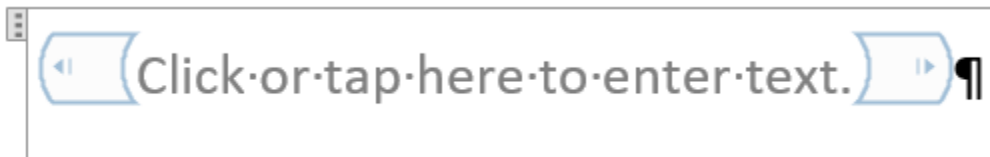
## Add a Hole Using a Rich Text Content Control

Add a hole to a Word template by using a rich text content control.

If you add a rich text content control with formatting symbols displayed and **Design Mode** enabled, you can tell whether you are creating a block or inline hole. For a block hole:

- The control bounding box includes a paragraph symbol.
- The control start and end tags contain left-pointing and right-pointing triangles, respectively. To see the triangles, zoom in on the Word document by opening the **View** tab and clicking **Zoom**.

Here is an example of a rich text control for a block hole:



For an inline hole:

- The control bounding box does not include a paragraph symbol.
- The control start and end tags do not include triangles.


Here is an example of a rich text control for an inline hole:



The procedure that you use to add a rich text content control depends on whether you are adding an inline or block hole and whether you want to add the hole in the document body or in a table cell.

- “Add an Inline Hole in a Document Body Paragraph” on page 13-122
- “Add a Block Hole in the Document Body” on page 13-122
- “Add an Inline Hole in a Table Entry” on page 13-123
- “Add a Block Hole in a Table Entry” on page 13-124

### Add an Inline Hole in a Document Body Paragraph

- 1 On the **Developer** tab, make sure that **Design Mode** is enabled. In this mode, when you create a hole, you can see the hole marks with the title tag.
- 2 Click in a paragraph where you want to create an inline hole. Add several blank spaces in front of the paragraph mark to make sure that you create an inline hole, not a block hole.
- 3 Click in front of the spaces.
- 4 Click the **Rich Text Content Control** button . A rich text content control appears at the insertion point.




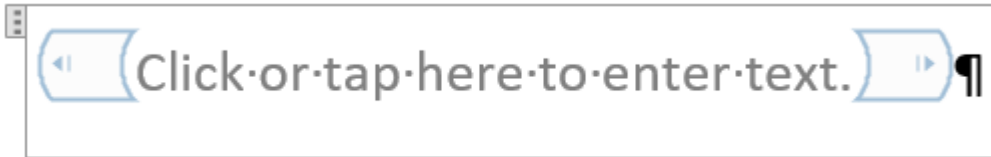
You can tell that you are creating an inline hole because the bounding box does not include the paragraph mark and the start and end tags do not include triangles.

- 5 Click the hole and then click **Properties**.
- 6 In the **Content Control Properties** dialog box:
  - In the **Title** field, enter an ID for the hole. Use this value to identify the hole in your report generation program or custom reporter class.
  - In the **Tag** field, enter `Hole`.
  - Optionally, set the default style for text that fills the hole. See “Specify a Default Style for Text That Fills a Hole” on page 13-124.
- 7 Click **OK**.
- 8 Delete the spaces that you added in front of the paragraph.

### Add a Block Hole in the Document Body

- 1 On the **Developer** tab, make sure that **Design Mode** is enabled. In this mode, when you create a hole, you can see the hole marks with the title tag.


- 2 Create an empty paragraph where you want to create a block hole. Creating a paragraph ensures that you create a block hole and not an inline hole. If you are at the end of a document, create an empty paragraph before the existing empty paragraph.
- 3 Select the empty paragraph that you added.
- 4 Click the **Rich Text Content Control** button . A rich text content control appears.



You can tell that you are creating a block hole because the bounding box includes the paragraph mark and the start and end tags include triangles.

- 5 Click the hole and then click **Properties**.
- 6 In the **Content Control Properties** dialog box:
  - In the **Title** field, enter an ID for the hole. Use this value to identify the hole in your report generation program or custom reporter class.
  - In the **Tag** field, enter `Hole`.
  - Optionally, set the default style for text that fills the hole. See “Specify a Default Style for Text That Fills a Hole” on page 13-124.
- 7 Click **OK**.

### Add an Inline Hole in a Table Entry

- 1 On the **Developer** tab, make sure that **Design Mode** is enabled. In this mode, when you create a hole, you can see the hole marks with the title tag.
- 2 To add an inline hole to a table entry, you must add the hole inside a paragraph in the entry. Every Word table entry is a paragraph, however, the paragraph mark is not visible. To ensure that you add the hole in the paragraph and that you do not include the default table cell in the hole, add a few spaces to the empty table entry.
- 3 Click in front of the spaces.
- 4 Click the **Rich Text Content Control** button . A rich text content control appears at the insertion point.



You can tell that you are creating an inline hole because the hole bounding box does not include a paragraph or table entry mark and the start and end tags do not include triangles.


- 5 Click the hole and then, click **Properties**.
- 6 In the **Content Control Properties** dialog box:
  - In the **Title** field, enter an ID for the hole. Use this value to identify the hole in your report generation program or custom reporter class.

- In the **Tag** field, enter Hole.
- Optionally, set the default style for text that fills the hole. See “Specify a Default Style for Text That Fills a Hole” on page 13-124.

7 Click **OK**.

8 Delete the spaces that you added.

### Add a Block Hole in a Table Entry

- 1 On the **Developer** tab, make sure that **Design Mode** is enabled. In this mode, when you create a hole, you can see the hole marks with the title tag.
- 2 To ensure that you create a hole that includes a paragraph and does not include the default table cell, add an empty paragraph to the table entry.
- 3 Select the empty paragraph that you added.
- 4 Click the **Rich Text Content Control** button . A rich text content control appears at the insertion point.



You can tell that you are creating a block hole because the hole bounding box includes the paragraph mark and the start and end tags include triangles. You can tell that the hole does not include the default cell because the bounding box does not include the table entry mark.

5 Click the hole and then click **Properties**.

6 In the **Content Control Properties** dialog box:

- In the **Title** field, enter an ID for the hole. Use this value to identify the hole in your report generation program or custom reporter class.
- In the **Tag** field, enter Hole.
- Optionally, set the default style for text that fills the hole. See “Specify a Default Style for Text That Fills a Hole” on page 13-124.

7 Click **OK**.

---

**Note** Block holes in a table do not support DocumentPart elements.

---

### Specify a Default Style for Text That Fills a Hole

You can specify a default style for text that fills a hole. Text formatting that you specify in your report generation program overrides the default style. See “Report Formatting Approaches” on page 13-17. The default text style applies to text that fills an inline hole and to text or a paragraph that fills a block hole. It does not apply to text in other report elements, such as tables, that can fill a block hole.

To specify the default style for text that fills a hole, in the **Content Control Properties** dialog box:



- 1 Select the **Use a style to format text typed into the empty control** check box.
- 2 Select the style in the **Style** list or click **New Style** to create a style.

For text in a paragraph, make sure that the style applies to a paragraph. If you create a style, in the **Create New Style from Formatting** dialog box, for **Style type**, select **Linked** (paragraph and character).

## See Also

### Related Examples

- “Form-Based Reporting” on page 13-24
- “Create a Microsoft Word Template” on page 13-119
- “Modify Styles in a Microsoft Word Template” on page 13-126
- “Fill the Blanks in a Report Form” on page 13-25
- “Create an HTML or PDF Template” on page 13-130

## Modify Styles in a Microsoft Word Template

### In this section...

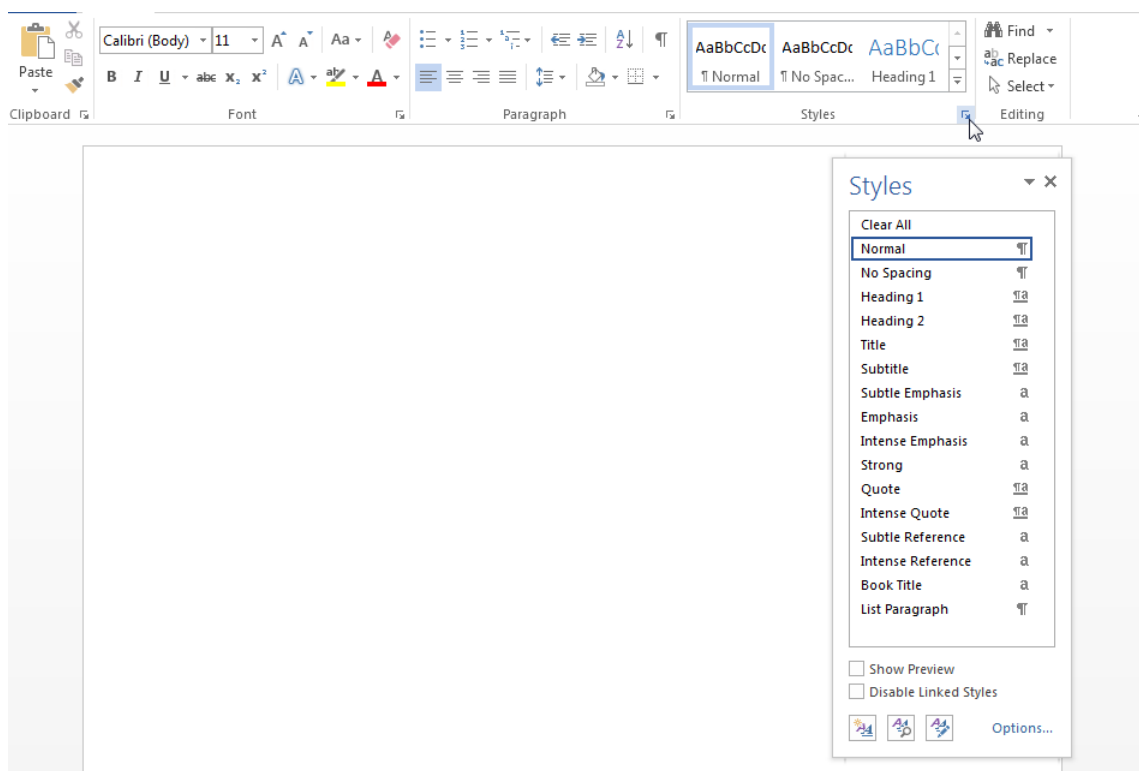
“Edit Styles in a Word Template” on page 13-126

“Add Styles to a Word Template” on page 13-127

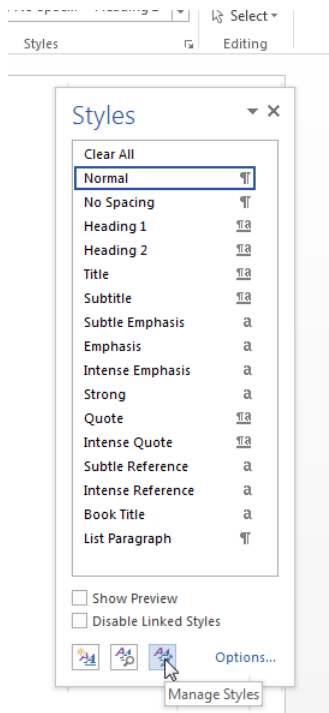
### Edit Styles in a Word Template

You can modify or add format styles in a Word template.

- 1 In your Word template, open the Styles pane.



- 2 In the Styles pane, click the **Manage Styles** button.



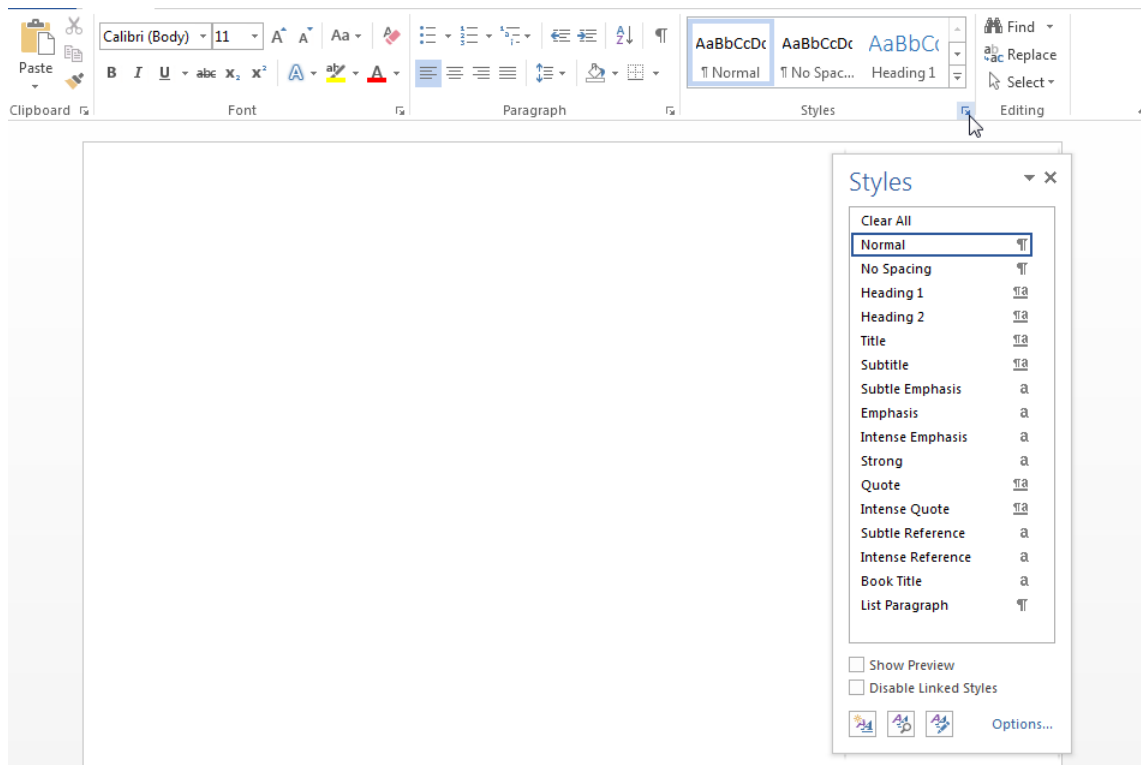
- 3 In the Manage Styles dialog box, click **Modify**.
- 4 In the Modify Styles dialog box, change any of the style definitions. For example, change the font family, font size, or indentation. When you have finished with your changes, click **OK**, and then close the Manage Styles dialog box.
- 5 Save and close the template.

For more information about using Word styles, see the Microsoft Word documentation.

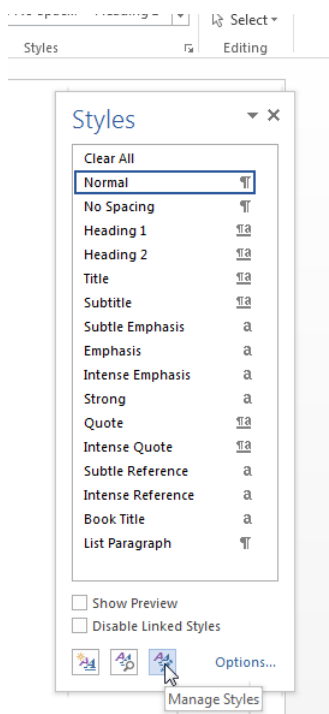
## Add Styles to a Word Template

- 1 In your Word template, open the Styles pane.

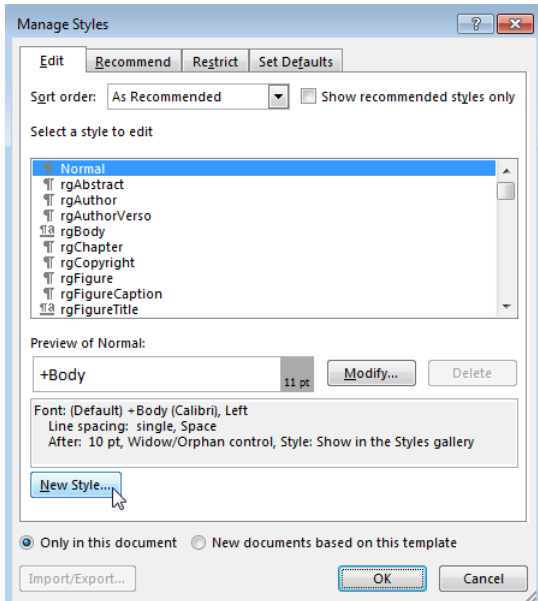
## 13 Create a Report Program



- 2 In the Styles pane, click the **Manage Styles** button.



- 3 Optionally, select an existing style to use as a starting point for the new style.
- 4 Click **New Style**.



- 5 Specify a name for the new style and define the style characteristics. To save the new style definition, click **OK** and close the dialog box.
- 6 Save and close the template.

## See Also

## Related Examples

- “Add Holes in a Microsoft Word Template” on page 13-120
- “Create an HTML or PDF Template” on page 13-130

## Create an HTML or PDF Template

Use one of these approaches to create an HTML or PDF template for generating a report.

- Use `mreportgen.dom.Document.createTemplate` to create a copy of the DOM API default template that you can then customize. For example:

```
mreportgen.dom.Document.createTemplate('mytemplate','html');
```

or, for a single-file HTML template,

```
mreportgen.dom.Document.createTemplate('mytemplate','html-file');
```

or, for PDF,

```
mreportgen.dom.Document.createTemplate('mytemplate','pdf');
```

- Create a template from scratch.

## Edit an HTML or PDF Template

A single-file HTML template embeds style sheets and images as HTML elements in the HTML document. An HTML template can be in a single file, with an `.html` extension, or in a zipped template package, with an `.htmlx` extension. PDF templates are packaged in a zipped template package, but use a `.pdfx` extension. To edit a single-file HTML template, open the `.html` file. To edit one of the packaged templates, unzip it to a folder using the `unzipTemplate` function. For example, to unzip an HTML template called `mytemplate` in the current folder:

```
unzipTemplate('mytemplate')
```

Using the `.htmlx` extension is optional for packaged HTML templates. However, to unzip a PDF template, you must use the template extension, for example:

```
unzipTemplate('mytemplate.pdfx')
```

After you unzip the template, you can edit the `.css` and `.html` files using a text editor or an HTML editor. To learn more templates, see “Templates for DOM API Report Programs” on page 13-22.

To repackage a template after you edit it, use the `zipTemplate` function. For example, package the template stored in a subfolder in the current folder named `mytemplate`:

```
zipTemplate('mytemplate.htmlx')
```

For PDF, use the `.pdfx` extension:

```
zipTemplate('mytemplate.pdfx')
```

If you want to work with your template in a location other than the current folder, you can specify a path with the `unzipTemplate` and `zipTemplate` functions.

## See Also

### Functions

`zipTemplate` | `unzipTemplate`

## **Classes**

`m\reportgen.dom.Document`

## **Related Examples**

- “Create an HTML Document Part Template Library” on page 13-30
- “Create a PDF Document Part Template Library” on page 13-32
- “Add Holes in HTML and PDF Templates” on page 13-132
- “Modify Styles in HTML Templates” on page 13-137
- “Modify Styles in PDF Templates” on page 13-138
- “Create a Microsoft Word Template” on page 13-119

## Add Holes in HTML and PDF Templates

Template holes are places in a template that a report program fills with generated content, supporting a form-based report.

### Types of Holes

You can create inline and block holes.

- An inline hole is for document objects that you can append to a paragraph: `Text`, `Image`, `LinkTarget`, `ExternalLink`, `InternalLink`, `CharEntity`, and `AutoNumber` objects.
- A block hole can contain a `Paragraph`, `Table`, `OrderedList`, `UnorderedList`, `DocumentPart`, and `Group`.

### Create a Hole

Use the same code to create a hole for inline and block holes. To create an inline hole, add the `<hole>` element to a paragraph. Create a block hole without a paragraph as its parent.

- 1 Unzip the template using the `unzipTemplate` command.
- 2 Open the `root.html` or `docpart_templates.html` file in an HTML or text editor.
- 3 Add code in any of these forms:

```
<hole id="HOLEID" default-style-name="STYLE_NAME">DESCRIPTION</hole>
```

```
<hole id="HOLEID" default-style-name="STYLE_NAME" />
```

```
<hole id="HOLEID" />
```

- Replace `HOLEID` with the hole identifier. If you need to get a hole ID or refer to a hole by ID in your report program, use this ID.
- Replace `STYLE_NAME` with the name of a default style to use for formatting the object appended to the hole. If you use this attribute, define the style in the template style sheet. Report generation uses this style if you do not specify one in your report program.

For inline holes, use a `span` element to define the default style, i.e., `span.STYLE_NAME`. For block holes, use the associated paragraph type, such as `p.STYLE_NAME` or `h1.STYLE_NAME`.

- Replace `DESCRIPTION` with text that describes the purpose of the hole.
- 4 Zip the template using the `zipTemplate` command.

### See Also

#### Functions

`zipTemplate` | `unzipTemplate`

### Related Examples

- “PDF and HTML Document Parts and Holes” on page 13-134
- “Simplify Filling in Forms” on page 13-37



- “Fill the Blanks in a Report Form” on page 13-25
- “Modify Styles in HTML Templates” on page 13-137
- “Modify Styles in PDF Templates” on page 13-138
- “Create a Microsoft Word Template” on page 13-119

## PDF and HTML Document Parts and Holes

This example shows how to:

- Define a document part template that has holes.
- Insert a document part into the report programmatically and fill holes.
- Insert a TOC document part.

This example uses a PDF template and report. However, you can use this same process for HTML reports. Replace the document type information with the corresponding HTML information throughout the example.

### Add Template to PDF Document Part Template Library

In this example, start with the default PDF template package.

- 1 Create a copy of the default template package.

```
m1reportgen.dom.Document.createTemplate('myPDFtemplate','pdf');
```

- 2 Unzip the template package.

```
unzipTemplate('myPDFtemplate.pdf');
```

- 3 In the current folder, open the unzipped template folder myPDFtemplate. Open docpart\_templates.html in an HTML or text editor.

The `dplibrary` element defines a document part library. The `dptemplate` element defines each document part template. This document part library has two document part templates:

- `rgChapter`, which defines a part template for chapters
- `ReportTOC`, which defines the table of contents

```
<html>
<body>
  <dplibrary>
    <dptemplate name="rgChapter">
      <h1 class="rgChapterTitle">
        <hole id="rgChapterTitlePrefix" default-style-name="rgChapterTitlePrefix" />
        <span> </span>
        <hole id="rgChapterTitleNumber" default-style-name="rgChapterTitleNumber" />
        <span>. </span>
        <hole id="rgChapterTitleText" default-style-name="rgChapterTitleText" />
      </h1>
      <hole id="rgChapterContent"/>
    </dptemplate>
    <dptemplate name="ReportTOC">
      <TOC number-of-levels="3" leader-pattern="dots" />
    </dptemplate>
  </dplibrary>
</body>
</html>
```

- 4 Create a document part template named `Author`. A document part can contain any combination of fixed text and holes. This document part template contains the fixed text `Author` and a hole for the author name.

```
<dptemplate name="Author">
  <p class="Author">
```

```
<span>Author: </span><hole id="AuthorName" />
</p>
```

```
</dptemplate>
```

- 5 Add the new document part template to the library. Because you refer to the document part by name when you call it from the API, you can put the templates in any order within the library. Use a unique name for each document part template.

```
<dplibrary>
  <dptemplate name="rgChapter">
    <h1 class="rgChapterTitle">
      <hole id="rgChapterTitlePrefix" default-style-name="rgChapterTitlePrefix" />
      <span> </span>
      <hole id="rgChapterTitleNumber" default-style-name="rgChapterTitleNumber" />
      <span>. </span>
      <hole id="rgChapterTitleText" default-style-name="rgChapterTitleText" />
    </h1>
    <hole id="rgChapterContent"/>
  </dptemplate>
  <dptemplate name="ReportTOC">
    <TOC number-of-levels ="3" leader-pattern="dots" />
  </dptemplate>
  <dptemplate name="Author">
    <p class="Author">
      <span>Author: </span><hole id="AuthorName" />
    </p>
  </dptemplate>
</dplibrary>
```

- 6 Repackage the template to a new template called `myPDFtemplate2.pdfctx`.

```
zipTemplate('myPDFtemplate2.pdfctx', 'myPDFtemplate');
```

## Use the Document Part Template in a Report Program

Use `mlreportgen.dom.DocumentPart` to use the document part template. You need:

- The name of the template package that contains the document part. In this example, the template package name is `myPDFtemplate2`.
- The names of the document part templates to call and the order of any holes you want to fill. In this example, you call:
  - The document part template `rgChapter` and fill the first three holes in the order of prefix, number, and title
  - The `ReportTOC` document part template, which inserts a table of contents
  - The `Author` document part template you created and fill its one hole

```
import mlreportgen.dom.*
d = Document('myDocPartEx', 'pdf', 'myPDFtemplate2');
open(d);

% Assign the rgChapter document part template to the variable dp
dp = DocumentPart(d, 'rgChapter');

% Move to each hole in this document part and append content
moveToNextHole(dp);
append(dp, 'Chapter');
moveToNextHole(dp);
append(dp, '5');
moveToNextHole(dp);
```

```
append(dp, 'Creating Document Part Templates');

% Append this document part to the document
append(d, dp);

% Append the document part ReportTOC to the document
append(d, DocumentPart(d, 'ReportTOC'));

% You can append any allowable object between document parts or holes
append(d, Paragraph('Append any allowable object or a document part.'));
append(d, Paragraph('Append a document part next:'));

% Assign the Author document part template to the variable dp2
dp2 = DocumentPart(d, 'Author');

% Move to the next hole and fill it
% Append the document part to the document
moveToNextHole(dp2);
append(dp2, 'Charles Brown');
append(d, dp2);

close(d);
rptview(d.OutputPath);
```

The Author document part template includes fixed text that precedes the hole. `moveToNextHole` appends any fixed content in the template between the previous hole (or the beginning of the document part) and the current hole to the document.

## See Also

`moveToNextHole`

## Related Examples

- “Create a PDF Document Part Template Library” on page 13-32
- “Add Holes in HTML and PDF Templates” on page 13-132
- “Create an HTML Document Part Template Library” on page 13-30

## Modify Styles in HTML Templates

You can customize or add format styles in the CSS files in your HTML template. You can use any CSS property in your style sheets.

- 1 In your unzipped template, navigate to `TEMPLATEROOT/Stylesheet`.
- 2 In a text or HTML editor, edit the `.css` file for the styles that you want to create or modify.

For information about editing a cascading style sheet, see documentation such as the W3Schools.com CSS tutorial.

- 3 Save the style sheet.

### See Also

#### Related Examples

- “Add Holes in HTML and PDF Templates” on page 13-132
- “Create a Microsoft Word Template” on page 13-119
- “Modify Styles in PDF Templates” on page 13-138

## Modify Styles in PDF Templates

<b>In this section...</b>
---------------------------

“PDF Style Sheets” on page 13-138
-----------------------------------

“Hyphenation Styles in PDF Templates” on page 13-141
--

You can customize or add format styles in your PDF template using this workflow. For information on properties you can use in PDF style sheets, see “PDF Style Sheets” on page 13-138.

- 1 In your unzipped template, navigate to `TEMPLATEROOT/Stylesheet`.
- 2 In a text or HTML editor, edit the cascading style sheet (`.css`) file for the styles you want to create or modify.

For information about editing a cascading style sheet, see documentation such as the W3Schools.com CSS tutorial.

- 3 Save the style sheet.

### PDF Style Sheets

Use the style sheet to define global styles, that is, the appearance of your template elements. You define PDF styles primarily using a subset of cascading style sheet (CSS) formats. You can also use XSL formatting objects (FO) to format elements in a PDF template. However, to simplify and streamline your code, use FO only for properties you cannot define using CSS.

Using a style sheet for the default formats simplifies your program. You also make fewer updates when your formatting changes. Format elements in your DOM program (for example, by using an object’s `Style` property) when you want to override the default format for an instance.

You can use a subset of CSS formats and this subset of selectors and selector combinators:

- Universal selector (\*)
- Type selector (for example, `p` or `span`)
- Class selector (for example, `p.MyPara`)
- Descendant combinator (space)
- Child combinator (`>`)
- Adjacent sibling combinator (`+`)
- General sibling combinator (`~`)

---

**Note** You can use the generalized sibling (`~`) and adjacent sibling (`+`) selectors only when creating the report in memory. If you are using streaming mode, do not use these selectors.

---

These CSS formats are supported:

- `background-color`
- `border`
- `border-bottom`

- border-bottom-color
- border-bottom-style
- border-bottom-width
- border-color
- border-left
- border-left-color
- border-left-style
- border-left-width
- border-right
- border-right-color
- border-right-style
- border-right-width
- border-style
- border-top
- border-top-color
- border-top-style
- border-top-width
- border-width
- color
- counter-increment
- counter-reset
- font-family
- font-size
- font-style
- font-weight
- height
- line-height
- list-style-type
- margin
- margin-bottom
- margin-left
- margin-right
- margin-top
- padding
- padding-bottom
- padding-left
- padding-right
- padding-top
- text-align

- text-decoration
- text-indent
- vertical-align
- white-space
- width

These custom CSS formats are supported:

- page-border
- page-border-color
- page-border-width
- page-border-style
- page-border-margin
- page-border-bottom
- page-border-bottom-color
- page-border-bottom-width
- page-border-bottom-style
- page-border-bottom-margin
- page-border-left
- page-border-left-color
- page-border-left-width
- page-border-left-style
- page-border-left-margin
- page-border-right
- page-border-right-color
- page-border-right-width
- page-border-right-style
- page-border-right-margin
- page-border-top
- page-border-top-color
- page-border-top-width
- page-border-top-style
- page-border-top-margin
- page-border-surround-header
- page-border-surround-footer
- page-margin
- page-margin-top
- page-margin-left
- page-margin-bottom
- page-margin-right



- `page-margin-header`
- `page-margin-footer`
- `page-margin-gutter`
- `page-size`
- `page-width`
- `page-height`
- `page-orientation`
- `halign`
- `valign`

For information about FO, see <https://www.w3.org/TR/xsl11/#fo-section>.

## Hyphenation Styles in PDF Templates

You can enable or disable hyphenation for paragraph and table cell styles that you define. You can also specify a hyphenation character. Alternatively, you can specify hyphenation on an instance of a `<p>` or `<td>` element.

### Specify Hyphenation for PDF Styles

You can specify hyphenation when you define a paragraph or table cell style. Use the `hyphenation` style with the name of the hyphenation character (`hyphen` or `space`), or use `none` to turn hyphenation off. If your style does not specify hyphenation, hyphenation is off by default for paragraphs and on by default for table cells, using a space character. These examples show the possible values for defining hyphenation in your CSS:

- `p.Style1 { hyphenation: hyphen; }`
- `td.Style2 { hyphenation: space; }`
- `p.SentenceStyle { hyphenation: none; }`

### Specify Hyphenation in PDF Tags

You can use a hyphenation value with the `style` attribute of paragraph and table cell styles. Use the value in the form `hyphenation:hyphenStyle;`, where `hyphenStyle` is `none`, `hyphen`, or `space`. For example:

```
<p style="hyphenation:hyphen;">Paragraph text</p>
```

If you do not specify the value, or hyphenation is not specified in the CSS, the default is no hyphenation for paragraphs and table cells.

## See Also

[mlreportgen.dom.Hyphenation](#) | [mlreportgen.dom.PDFPageLayout](#) | [mlreportgen.dom.TableEntry](#)

## Related Examples

- “Add Holes in HTML and PDF Templates” on page 13-132
- “Create a Microsoft Word Template” on page 13-119

- “Modify Styles in HTML Templates” on page 13-137

### **External Websites**

- <https://www.w3.org/TR/xsl11/#fo-section>
- [www.w3schools.com/cssref](http://www.w3schools.com/cssref)

## Create Chapters

You can add chapters to your report using the `mlreportgen.report.Chapter` class. It is much easier and more efficient to use this class rather than using DOM objects to create a chapter. The `Chapter` class inherits from `mlreportgen.report.Section`. The `Chapter` class automatically adds a formatted chapter into your report. The default formatting is portrait orientation with a header and a footer. The footer includes the page number. You can override the layout of the chapter.

For information and examples, see `mlreportgen.report.Chapter`.

This example shows code that creates a title page and sets its page orientation to landscape.

```
import mlreportgen.report.*
import mlreportgen.dom.*

rpt = Report('My Report','pdf');

add(rpt,TitlePage('Title','My Report'));

chapter = Chapter('Images');
chapter.Layout.Landscape = true;
add(chapter,Section('Title','Boeing 747', ...
    'Content',Image(which('b747.jpg'))));

add(rpt,chapter);
close(rpt);
rptview(rpt);
```



## Create Page Layout Sections

You can add sections to a report using the `mlreportgen.report.Section` class. This predefined class automatically adds a formatted section into your report. The default formatting is portrait orientation with a default margins and a page number in the footer. You can override the layout and contents of the section. It is much easier and more efficient to use this class rather than using DOM objects to create a section. For information and examples, see `mlreportgen.report.Section`

You can also use DOM objects to create sections. You can divide a Word or PDF document into sections, each with its own page layout. Page layout includes page margins, page orientation, and headers and footers.

### Define Page Layouts in a Word Template

Every Word template has at least one page layout section. You can use Word to create as many additional sections as you need. For example, in the main template for a report, you can create sections for your report's title page, table of contents, and chapters. See the Word documentation for information on how to create page layout sections in a Word template.

### Define Page Layouts in a PDF Template

You define page layouts in a PDF template using a `<layout>` element. You can use the `<layout>` element in the main template (`root.html`), and in document part templates.

You can use these attributes with the `<layout>` element.

<code>style</code>	<code>page-margin: top left bottom right header footer gutter; page-size: height width orientation</code>
<code>first-page-number</code>	Number of first page in the layout
<code>page-number-format</code>	n or N for numeric, a, A, i, I
<code>section-break</code>	Where to start section for this layout: Odd Page, Even Page, or Next Page

For example, this element defines a layout with:

- Top, bottom, left, and right margins of 1 inch
- Header and footer heights of 0.5 inches
- Gutter size (space for binding pages) of 0
- 8.5-inch by 11-inch page size in portrait orientation

```
<layout style="page-margin: 1in 1in 1in 1in 0.5in 0.5in 0in;
    page-size: 8.5in 11in portrait" />
```

This `<layout>` element includes a page footer. The page footer `DefaultPageFooter` must be defined in a document part template.

```
<layout style="page-margin: 1in 1in 1in 1in 0.5in 0.5in 0in;
    page-size: 8.5in 11in portrait">
    <pfooter type="default" template-name="DefaultPageFooter" />
</layout>
```

You can create page layouts in document parts. For example, this code defines a document part template named `Chapter` that includes a page layout. The layout includes a page header and a page footer and specifies the format for the page number using the `<pnumber>` element. In this case, also define part templates for the page header and page footer elements. See “Use Page Headers and Footers in a Template” on page 13-148.

```
<dptemplate name="Chapter">
  <layout style="page-margin: 1in 1in 1in 1in 0.5in 0.5in 0in;
                page-size: 8.5in 11in portrait">
    <pheader type="default" template-name="MyPageHeader"/>
    <pfooter type="default" template-name="MyPageFooter"/>
    <pnumber format="1" />
  </layout>
  <!-- Define content for your layout here--fixed text and holes as needed -->
</dptemplate>
```

To use the layout, insert the document part into your report using your program. This code assumes that there is one hole in the document part `Chapter`. The document part uses the page layout definition you provided in the `Chapter` document part template.

```
import mlreportgen.dom.*
d = Document('myDocPartEx', 'pdf', 'mytemplate');
open(d);

% Assign the Chapter document part template to the variable dp
dp = DocumentPart(d, 'Chapter');

% Move to each hole in this document part and append content
moveToNextHole(dp);
append(dp, 'My text to fill hole');

% Append this document part to the document
append(d, dp);

close(d);
rptview(d.OutputPath);
```

## Watermarks in PDF Page Layouts

You can place a watermark in a PDF page layout. A watermark is an image that appears in the background of a page, such as the word `Draft` or `Confidential`. It runs behind the text on each page you apply it to. You can use any of these file types for the image: `.bmp`, `.jpg`, `.pdf`, `.png`, `.svg`, and `.tiff`.

Use `<watermark>` in a `<layout>` element. Specify the watermark as an image file stored in the template package. To store the image in the template package, unzip the template package, copy the image into the folder, and then zip the template again. For example:

- 1 Unzip the template.
 

```
unzipTemplate('MyTemplate.pdfctx');
```
- 2 Copy the watermark image into the folder `MyTemplate`. To keep your images organized, copy the image into the `images` folder.
- 3 Add the watermark element to a page layout in your template. For example, add the watermark to the default layout in `root.html`.

```
<layout style="page-margin: 1in 1in 1in 1in 0.5in 0.5in 0in;
                page-size: 8.5in 11in portrait" >
```

```
<watermark src="./images/myfile.png" width="6in" />
</layout>
```

- 4 Zip the template.

```
zipTemplate('MyTemplate.pdf', 'MyTemplate');
```

- 5 Delete the folder MyTemplate.
- 6 Create a report that uses this template using the DOM API, or create a form-based report in Report Explorer whose **PDF Page Layout** component uses this layout.

## Navigate Template-Defined Page Layouts

A document or document part's `CurrentPageLayout` property points to a page layout object that specifies the current section's page layout based on the document or document part's template. Each time you move to a new section (by moving to a hole at the beginning of the section), the DOM updates the `CurrentPageLayout` property to point to the page layout object that specifies the section's page layout properties. You can change a section's page layout by modifying the properties of the layout object or replacing the layout object with a new object.

For example, you can change the section's orientation or add page headers or footers. Make these changes before you add any content to the new section. When replacing the current layout object, use an `mlreportgen.dom.DOCXPageLayout` object for Word documents and `mlreportgen.dom.PDFPageLayout` for PDF documents.

## Override Template Page Layouts in Your Report Program

You can change the template-defined layout properties programmatically. For example, the page orientation of the DOM default Word template is portrait. This example changes the orientation to landscape to accommodate wide tables. The code swaps the height and width of the page to the new page orientation.

```
import mlreportgen.dom.*
rpt = Document('test', 'docx');
open(rpt);

sect = rpt.CurrentPageLayout;
pageSize = sect.PageSize;
pageSize.Orientation = 'landscape';

saveHeight = pageSize.Height;
pageSize.Height = pageSize.Width;
pageSize.Width = saveHeight;

table = append(rpt, magic(22));
table.Border = 'solid';
table.ColSep = 'solid';
table.RowSep = 'solid';

close(rpt);
rptview(rpt.OutputPath);
```

## Create Layouts Programmatically

You can append a `DOCXPageLayout` object (for Word documents) or a `PDFPageLayout` object (for PDF documents) to start a new page layout section programmatically. For DOCX reports, the `append` method can specify a paragraph to end the previous section.

```
append(rptObj, paraObj, LayoutObj)
```

If you do not specify a paragraph in your `append` method, the DOM API inserts an empty paragraph before starting the new section. This example uses the end paragraph syntax to avoid inserting an empty paragraph at the end of the previous section.

```
import mlreportgen.dom.*
rpt = Document('test', 'docx');

append(rpt, Heading(1, 'Magic Square Report', 'Heading 1'));

sect = DOCXPageLayout;
sect.PageSize.Orientation = 'landscape';
sect.PageSize.Height = '8.5in';
sect.PageSize.Width = '11in';
append(rpt, Paragraph('The next page shows a magic square.'), sect);

table = append(rpt, magic(22));
table.Border = 'solid';
table.ColSep = 'solid';
table.RowSep = 'solid';

close(rpt);
rptview(rpt.OutputPath);
```

## See Also

### Classes

`mlreportgen.dom.PDFPageLayout` | `mlreportgen.dom.DOCXPageLayout` |  
`mlreportgen.dom.PageSize` | `mlreportgen.dom.PageMargins` |  
`mlreportgen.dom.PageBorder`

## Related Examples

- “Create a Microsoft Word Template” on page 13-119

## Create Page Footers and Headers

You can create page headers and page footers in Word and PDF reports. You can create page headers and page footers in each layout for each of these types of pages:

- The first page of the section
- Even pages
- Odd pages, which include the first page if you do not specify a first-page header or footer

You can create report page headers and footers programmatically or in the template to use with the report. You can append content to the footers.

When you open a report, the DOM API:

- 1 Reads the headers and footers from the template and converts them to PDF or DOCX `PageHeader` and `PageFooter` objects
- 2 Associates the headers and footer objects with the DOCX or PDF `PageLayout` object that defines the properties of the section that contains the headers and footers
- 3 Adds the headers and footers to your report as your code navigates the sections defined by the template

As your report program navigates the sections, it can append content to the template-defined headers and footers.

## Use Page Headers and Footers in a Template

You can insert page headers and footers in the main template or in a document part template. The approach differs for Word and for PDF.

### Page Headers and Footers in a Word Template

Every page in a Word document has a header and footer that you can edit. To enable editing mode, double-click the header or footer area. Alternatively, on the Word **Insert** tab, in the **Header & Footer** section, click the **Header** or **Footer** button arrow. From the menu, select the corresponding **Edit** command. When you have finished editing the header or footer, on the **Header & Footer Tools Design** tab, click **Close Header and Footer**.

In editing mode, you can modify the header or footer by:

- Inserting text, holes, page numbers, and images
- Formatting the items you add, for example, by specifying the page number type
- Resizing the header or footer
- Specifying a different header or footer for the first page, odd pages, and even pages
- Inserting Word fields

The fields mechanism helps you to generate header or footer content that varies from page to page. To see the fields you can insert, click the **Explore Quick Parts** button and select **Field**. The `StyleRef` field is useful for inserting chapter or section titles in the footer. See “Create Running Page Headers and Footers” on page 13-151.



For details about working with Word page headers and footers, see <https://support.microsoft.com/en-us/word>.

You can modify the page headers and footers directly in the main template. To add a page header or footer in a document part template, modify the page header and footer as you want. Select the entire page using **CTRL+A** before you save the part to the Quick Parts Gallery. For details on adding and modifying document part templates, see “Create a Microsoft Word Document Part Template Library” on page 13-27.

You can insert a page number in a header or footer. On the **Header & Footer Design** tab, use the **Page Number** menu to insert a page number. To access formatting options, in the header or footer, right-click the page number and select **Format Page Numbers**.

### Page Headers and Footers in a PDF Template

Adding page headers and footers in a PDF template involves these steps:

- Add `<pheader>` and `<pfooter>` elements to a page layout that you define using the `<layout>` element. You can add the header and footer elements to the layout in the main template (`root.html`) or in a document part template.
- Define a document part template for each page header or footer style.

---

**Note** If you insert the header or footer into a layout only programmatically, you do not need to add the `<pfooter>` or `<pheader>` element to a template `<layout>` element.

---

The table shows the attributes that you can use with `<pheader>` and `<pfooter>`. These elements correspond with the DOM classes `mlreportgen.dom.PDFPageHeader` and `mlreportgen.dom.PDFPageFooter`.

Element	Attributes	Values
pheader	type	default, first, even
	template-name	Document part template that defines the header
pfooter	type	default, first, even
	template-name	Document part template that defines the footer

For example, this code defines a document part template `Chapter` that uses two page footers: one for odd pages and one for even pages. The page number format is Arabic numerals.

```
<dptemplate name="Chapter">
  <layout style="page-margin: 1in 1in 1in 1in 0.5in 0.5in 0in;
    page-size: 8.5in 11in portrait">
    <pfooter type="default" template-name="MyPageFooter"/>
    <pfooter type="even" template-name="MyEvenFooter"/>
    <pnumber format="1" />
  </layout>
  <!-- Define content for your chapter here--fixed text and holes as needed -->
</dptemplate>
```

Define the document part templates `MyPageFooter` and `MyEvenFooter` in the `docpart_templates.html` file. For example, define the page footers so that:

- All footers insert a page number. To insert a page number, use the `<page>` element.
- The odd page numbers align right. The default value for `type` on the `pfooter` element specifies first and odd pages.

- The even page numbers align left.

These document part templates define the page footers.

```
<dptemplate name="MyPageFooter">
  <p style="text-align:right;font-family:Arial,Helvetica,sans-serif;font-size:10pt">
    <page/></p>
</dptemplate>
<dptemplate name="MyEvenFooter">
  <p style="text-align:left;font-family:Arial,Helvetica,sans-serif;font-size:10pt">
    <page/></p>
</dptemplate>
```

These DOM API HTML elements are useful in headers and footers. For example, you can add page numbers to headers and footers in the form Page 1 of 3 using <page> and <numpages>. See `mlreportgen.dom.NumPages` for the equivalent programmatic approach. You can also generate content in the header or footer that changes based on the content of a specified element (style) on the page. See “Create Running Page Headers and Footers” on page 13-151.

Purpose	Element	Attributes	Values
Page number format (same as first-page-number and page-number-format on layout)	pnumber	format	n or N for numeric, a, A, i, I
		initial-value	The number for the first page in the layout that uses this element
Current page number	page	No attributes	n/a
Total number of pages in document	numpages	No attributes	n/a
Insert content of a heading or other style into a page header or footer (for running headers and footers)	styleref	No attributes	Inserts content of nearest h1 element.
		style-name or	Name of the style with content to insert in the header or footer, or
		outline-level	Outline level of style with content to insert in the header or footer

**Access Template-Defined Headers and Footers**

Use the `CurrentPageLayout` property of a `Document` or `DocumentPart` object to access the template-defined headers and footers for the current section of a document or document part.

The value of the `CurrentPageLayout` property is a `DOCXPageLayout` or `PDFPageLayout` object whose `PageHeaders` and `PageFooters` properties contain a cell array of objects corresponding to the template-defined headers and footers of the current section. Each cell array can contain up to three objects, depending on how many of the three types of headers and footers (first page, even page, odd page) you define for the section. The objects can appear in any order in the cell array. Thus, to access a header or footer of a particular type, search the cell array to find the one you want to access.

## Append Content to a Template-Defined Header or Footer

You can use the DOM API to append content to a template-defined header or footer that appears on every page in a section. To append content to a header or footer in the current section of a document or document part, first use the document or document part `CurrentPageLayout` property to access the DOCX or PDF `PageHeader` or `PageFooter` object. Then use the `append` method of a `PageHeader` or `PageFooter` object to append content.

Header and footer objects are a type of document part object. You can append any kind of content to a page header or footer that you can append to a document part, for example, paragraphs, images, and tables.

You can use holes in the header and footers of your main template to control the positioning of content that you append to the headers and footers. For example, this program appends today's date to a hole named `Date` on the first template-defined page header of the first section of a report. This example assumes that the Word template `MyReportTemplate` has one layout that defines a first page, odd page, and even page header and footer.

```
import mlreportgen.dom.*;
d = Document('MyReport', 'docx', 'MyReportTemplate');
open(d);

sect = d.CurrentPageLayout;

for i = 1:numel(sect.PageHeaders)
    if strcmpi(sect.PageHeaders(i).PageType, 'first')
        firstPageHeader = sect.PageHeaders(i);
        while ~strcmp(firstPageHeader.CurrentHoleId, '#end#')
            switch firstPageHeader.CurrentHoleId
                case 'Date'
                    append(firstPageHeader, date);
            end
            moveToNextHole(firstPageHeader);
        end
        break;
    end
end

close(d);
rptview(d.OutputPath);
```

## Create Running Page Headers and Footers

A running page header or footer contains content that varies from page to page based on context. For example, the name of the current chapter or section changes from page to page. You can insert the current content in a page header or footer.

You can create running page headers and footers programmatically or in a template.

### Create Running Page Headers and Footers in a Template

“Page Headers and Footers in a Word Template” on page 13-148 describes the general approach to editing page headers and footers in Word. To add running text, insert a `StyleRef` field. This field is equivalent to the DOM API `mlreportgen.dom.StyleRef` class. To insert this field in a Word template or document part template:

- 1 Open the header or footer for editing.
- 2 On the **Insert** tab, from the **Quick Parts** button menu, select **Field**.
- 3 In the Field dialog box, from the **Field names** list, select **StyleRef**. From the **Style name** list, select the name of the style that contains the text that you want to include in the running header or footer.

For example, select **Heading 1** to use the content of paragraphs formatted with the **Heading 1** style. Your report must create content that uses that style for the content to appear in the header or footer.

- 4 Click **OK**.

For PDF documents, to include running text, use a `<styleref>` element. Add code like this to your template's `docpart_templates.html` library file. The `<styleref>` element uses the `Heading1` object for the content by default.

```
<dptemplate name="RunningFooter">
  <p style="text-align:center;font-family:sans-serif;font-size:10pt">
    <styleref/>
  </p>
</dptemplate>
```

To see the effect, add the page footer in the `<layout>` element of your template's `root.html` file. You can insert it in any `<layout>` element your template defines.

```
<layout style="page-margin: 1in 1in 1in 1in 0.5in 0.5in 0in;
  page-size: 8.5in 11in portrait">
  <pfooter template-name="RunningFooter" />
</layout>
```

Use code that creates `Heading1` objects and calls your template to see the result. This code assumes that you defined the footer document part template in the template `RunFooters`.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'pdf', 'RunFooters');
open(d);

title = append(d, Paragraph('Document Title'));
title.Bold = true;
title.FontSize = '28pt';

h1 = append(d, Heading1('My First Chapter'));
p1 = append(d, Paragraph('Hello World'));

h2 = append(d, Heading1('My Second Chapter'));
h2.Style = {PageBreakBefore(true)};
p2 = append(d, Paragraph('Text for this chapter.));

close(d);
rptview(d.OutputPath);
```

To refer to the page footer programmatically, use code in this form. The first argument is the type of footer, the second is the template package, and the third is the document part template.

```
PDFPageFooter('default', 'RunFooters', 'RunningFooter');
```

This code creates the footer in the current page layout without relying on the template to insert the footer. It uses the template only for the definition of the document part template.

```
import mlreportgen.dom.*;
d = Document('mydoc', 'pdf', 'RunFooters');
open(d);

myfooter = PDFPageFooter('default', 'RunFooters', 'RunningFooter');
d.CurrentPageLayout.PageFooters = myfooter;

title = append(d, Paragraph('Document Title'));
title.Bold = true;
title.FontSize = '28pt';

h1 = append(d, Heading1('My First Chapter'));
p1 = append(d, Paragraph('Hello World'));

h2 = append(d, Heading1('My Second Chapter'));
h2.Style = {PageBreakBefore(true)};
p2 = append(d, Paragraph('Text for this chapter.));

close(d);
rptview(d.OutputPath);
```

### Create Running Page Headers and Footers Programmatically

The DOM API provides classes that help you to create running headers and footers programmatically in Word and PDF documents.

- To insert a chapter title in a page header or footer, see `mlreportgen.dom.StyleRef`.
- To work with page headers and footers, see `mlreportgen.dom.DOCXPageHeader`, `mlreportgen.dom.DOCXPageFooter`, `mlreportgen.dom.PDFPageHeader`, and `mlreportgen.dom.PDFPageFooter`.

### Create Page Headers and Footers Programmatically

Programmatically create a page header or footer in the current section of a report. You can use the same technique for PDF, using `PDFPageHeader` and `PDFPageFooter` in place of the corresponding DOCX parts.

- 1 Use the `DOCXPageHeader` or `DOCXPageFooter` constructor to create the desired type of page header or footer (first page, odd page, even page, or odd and even page) based on a template that defines template form (the fixed content and holes for variable content).
- 2 Fill the holes in the header or footer with content.
- 3 Insert the header or footer in the array of page headers or footers of the current `PageLayout` object.

This code creates a first page header from a template stored in the document part template library of a report.

```
import mlreportgen.dom.*;
d = Document('MyReport', 'docx', 'MyReportTemplate');
open(d);

pageHeaders(1) = DOCXPageHeader('first', d, 'FirstPageHeader');
```

```
while ~strcmp(pageHeaders(1).CurrentHoleId, '#end#')
    switch pageHeaders(1).CurrentHoleId
        case 'Date'
            append(pageHeaders(1), date);
        end
    moveToNextHole(pageHeaders(1));
end
```

```
d.CurrentPageLayout.PageHeaders = pageHeaders;
```

```
close(d);
rptview(d.OutputPath);
```

- To insert a page number, use an `mlreportgen.dom.Page` object.
- To insert a page number in the form Page [current page] of [total pages], see `mlreportgen.dom.NumPages`.
- To insert complex page numbers in a Word report, in the form [Chapter #]-[Current Page #], see “Add Complex Page Numbers in Microsoft Word” on page 13-155.

## See Also

### Functions

`mlreportgen.dom.Document.createTemplate`

### Classes

`mlreportgen.dom.DOCXPageHeader` | `mlreportgen.dom.DOCXPageFooter` |  
`mlreportgen.dom.DOCXPageLayout` | `mlreportgen.dom.PDFPageHeader` |  
`mlreportgen.dom.PDFPageFooter` | `mlreportgen.dom.PDFPageLayout` |  
`mlreportgen.dom.DocumentPart` | `mlreportgen.dom.Document` | `mlreportgen.dom.Page` |  
`mlreportgen.dom.NumPages` | `mlreportgen.dom.StyleRef`


## Related Examples

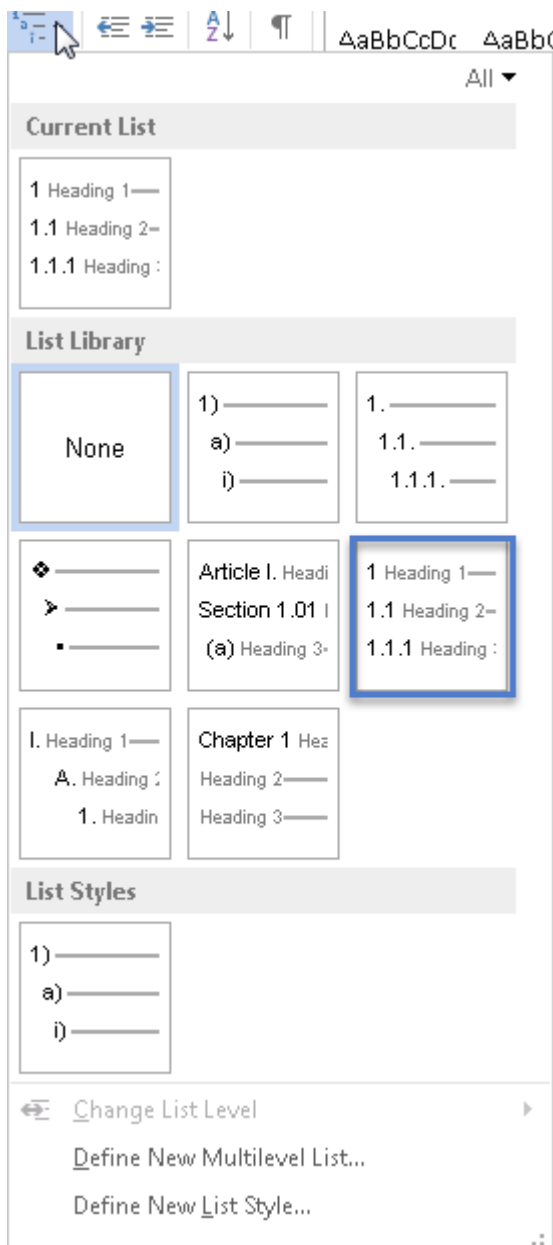
- “Create a Microsoft Word Template” on page 13-119
- “Create an HTML or PDF Template” on page 13-130
- “Add Complex Page Numbers in Microsoft Word” on page 13-155

## Add Complex Page Numbers in Microsoft Word

This example adds a complex page number to footers in Microsoft Word document. A complex number has the form [Chapter #][separator][Page#], for example, 7-1. You can add this type of number in a header or footer. You can do this using a template, by inserting a page number in a footer, and specifying the page number properties.

Whether you are using a template or a program, your template must use a multilevel list for the heading level that contains the chapter to reference. To create this type of list:

- 1 In your Word template, on the **Home** tab, click the **Multilevel List** button .
- 2 Select the numbered heading item.



- 3 Apply the Normal style to the paragraph.
- 4 Save and close the template.

You can then use a program like this one to use the complex page number. The `ChapterStartStyle` and `ChapterSeparator` properties on the `PageNumber` object specify to use heading level 1 for the chapter number and an en-dash as a separator.

```
import mlreportgen.dom.*;
d = Document('mypages', 'docx', 'numberChapters');

open(d);
layout = d.CurrentPageLayout;

% Page number formatting
pgnum = PageNumber(1, 'n');
pgnum.ChapterStartStyle = '1';
pgnum.ChapterSeparator = 'endash';

% Add page number object to page layout styles
layout.Style = {pgnum};
% layout.Style = [layout.Style {pgnum}];

% Create the footer object and insert a page number
myfooter = DOCXPageFooter();
para = Paragraph();
para.HAlign = 'center';
append(para, Page());
append(myfooter, para);

% Set PageFooters on the current layout to your footer object
layout.PageFooters = myfooter;

% Create content
for i=1:5
    title = append(d, Heading1(['Chapter' num2str(i)]));
    title.Style = {PageBreakBefore};
    for j=1:30
        append(d, 'This is the body of the chapter');
    end
end

close(d);
rptview(d.OutputPath);
```

---

**Tip** Create a page layout for each chapter to restart numbering the pages for each chapter at 1.

---

## See Also

`mlreportgen.dom.PageNumber` | `mlreportgen.dom.DOCXPageLayout` |  
`mlreportgen.dom.DOCXPageFooter`



# Functional Report

This example illustrates a functional approach to creating a report generator based on the DOM API. It uses the DOM API to create a MATLAB® function, `rptmagic`, that generates a PDF, HTML, or a Microsoft® Word report on a specified set of magic squares.

## Generate a Report

1. Open this example if it is not already open.
2. Generate an HTML, Word, or PDF report on four magic squares as follows:
  - Click to generate an HTML report:

```
rptmagic([10,20,40,75],'html',true)
```

- Click to generate a Word report:

```
rptmagic([10,20,40,75],'docx',true)
```

- Click to generate a PDF report:

```
rptmagic([10,20,40,75],'pdf',true)
```

## Edit the Example's Word Templates

This example uses two Microsoft Word templates to generate a Word report:

- A main template that defines the format and fixed content of the reports title, table-of-contents, and introductory chapter.
- A chapter template that defines the layout and fixed content of the chapters containing magic squares. The magic square chapter template is stored in the Quick Parts gallery of the main template.

You can edit these templates to change the appearance of a generated report:

1. Open this example if it is not already open.
2. Click `winopen('magic_squares_docx.dotx')` to open the template in Word
3. Edit the template(s) in Word.
4. Save the template(s).
5. Click `rptmagic([10,20,40,75],'docx',true)` to generate a report based on the modified template(s).

## Edit the Example's HTML Templates

This example uses two HTML templates to generate an HTML report:

- A main template that defines the format and fixed content of the reports title, table-of-contents, and introductory chapter.
- A chapter template that defines the layout and fixed content of the chapters containing magic squares. The magic square chapter template resides in the main template's document part template library.

To edit the templates:

1. Click `unzipTemplate('magic_squares_html.htmxtx')` to unzip the template.
2. Click `edit('magic_squares_html/root.html')` to open the template root document in the MATLAB editor.
3. Click `edit('magic_squares_html/docpart_templates.html')` to open the document part templates library document in the MATLAB editor.
4. Click `edit('magic_squares_html/stylesheets/root.css')` to open the report style sheet in the MATLAB editor.
4. Edit the template's files in the MATLAB editor.
5. Save the templates files.
6. Click `zipTemplate('magic_squares_html.htmxtx')` to rezip the template.
7. Click `rptmagic([10,20,40,75],'html',true)` to generate a report based on the modified template.

### **Edit the Example's PDF Templates**

This example uses two PDF templates to generate a PDF report:

- A main template that defines the format and fixed content of the reports title, table-of-contents, and introductory chapter.
- A chapter template that defines the layout and fixed content of the chapters containing magic squares. The magic square chapter template resides in the main template's document part template library.

To edit the PDF templates:

1. Click `unzipTemplate('magic_squares_pdf.pdfctx')` to unzip the template.
2. Click `edit('magic_squares_pdf/root.html')` to open the template root document in the MATLAB editor.
3. Click `edit('magic_squares_pdf/docpart_templates.html')` to open the report style sheet in the MATLAB editor.
4. Click `edit('magic_squares_pdf/stylesheets/root.css')` to open the document part templates library document in the MATLAB editor.
5. Edit the template files in the MATLAB editor.
6. Save the files.
7. Click `zipTemplate('magic_squares_pdf.pdfctx')` to rezip the template.
8. Click `rptmagic([10,20,40,75],'pdf',true)` to generate a report based on the modified template.

### **Edit the Example Code**

1. Open this example if it is not already open.

2. Click `edit('rptmagic.m')` to open `rptmagic.m` in the MATLAB editor.
3. Edit `rptmagic.m`.
4. Save `rptmagic.m`.
6. Rerun `rptmagic.m` to see the results of your changes.

### **See Also**

`mlreportgen.dom.Document` | `mlreportgen.dom.DocumentPart` | `moveToNextHole` | `unzipTemplate` | `zipTemplate`

### **More About**

- “Create a Report Program” on page 13-3
- “Form-Based Reporting” on page 13-24
- “Templates for DOM API Report Programs” on page 13-22

## Object-Oriented Report

This example illustrates an object-oriented approach to creating a report generator based on the DOM API. It uses the DOM API to create pair of MATLAB® classes, `MagicSquareReport` and `MagicSquareChapter`, that generate a PDF, Microsoft® Word, or HTML report on a set of magic squares.

### Generate a Report

1. Open this example if it is not already open.
2. Generate an HTML, Word, or PDF report on four magic squares as follows:

- Click to generate an HTML report:

```
rptmagicoo([10,20,40,75],'html',true)
```

- Click to generate a Word report:

```
rptmagicoo([10,20,40,75],'docx',true)
```

- Click to generate a PDF report:

```
rptmagicoo([10,20,40,75],'pdf',true)
```

### Edit the Example's Word Templates

This example uses two Microsoft Word templates to generate a Word report:

- A main template that defines the format and fixed content of the reports title, table-of-contents, and introductory chapter.
- A chapter template that defines the layout and fixed content of the chapters containing magic squares. The magic square chapter template is stored in the Quick Parts gallery of the main template.

You can edit these templates to change the appearance of a generated report:

1. Open this example (if it is not already open).
2. Click `winopen('magic_squares_docx.dotx')` to open the template in Word
3. Edit the template(s) in Word.
4. Save the template(s).
5. Click `rptmagicoo([10,20,40,75],'docx',true)` to generate a report based on the modified template(s).

### Edit the Example's HTML Templates

This example uses two HTML templates to generate an HTML report:

- A main template that defines the format and fixed content of the reports title, table-of-contents, and introductory chapter.
- A chapter template that defines the layout and fixed content of the chapters containing magic squares. The magic square chapter template resides in the main template's document part template library.

To edit the HTML templates:

1. Click `unzipTemplate('magic_squares_html.htmxtx')` to unzip the template.
2. Click `edit('magic_squares_html/root.html')` to open the template root document in the MATLAB editor.
3. Click `edit('magic_squares_html/docpart_templates.html')` to open the document part templates library document in the MATLAB editor.
4. Click `edit('magic_squares_html/stylesheets/root.css')` to open the report style sheet in the MATLAB editor.
4. Edit the template's files in the MATLAB editor.
5. Save the templates files.
6. Click `zipTemplate('magic_squares_html.htmxtx')` to rezip the template.
7. Click `rptmagic([10,20,40,75],'html',true)` to generate a report based on the modified template.

### **Edit the Example's PDF Templates**

This example uses two PDF templates to generate a PDF report:

- A main template that defines the format and fixed content of the reports title, table-of-contents, and introductory chapter.
- A chapter template that defines the layout and fixed content of the chapters containing magic squares. The magic square chapter template resides in the main template's document part template library.

To edit the PDF templates:

1. Click `unzipTemplate('magic_squares_pdf.pdfctx')` to unzip the template.
2. Click `edit('magic_squares_pdf/root.html')` to open the template root document in the MATLAB editor.
3. Click `edit('magic_squares_pdf/docpart_templates.html')` to open the report style sheet in the MATLAB editor.
4. Click `edit('magic_squares_pdf/stylesheets/root.css')` to open the document part templates library document in the MATLAB editor.
5. Edit the template files in the MATLAB editor.
6. Save the files.
7. Click `zipTemplate('magic_squares_pdf.pdfctx')` to rezip the template.
8. Click `rptmagicoo([10,20,40,75],'pdf',true)` to generate a report based on the modified template.

### **Edit the Example Code**

1. Open this example if it is not already open.

2. Click `edit('rptmagicoo.m')` to open `rptmagicoo.m` in the MATLAB editor.
3. Edit `rptmagicoo.m`.
4. Save `rptmagicoo.m`.
6. Rerun `rptmagicoo.m` to see the results of your changes.

### See Also

`mlreportgen.dom.Document` | `mlreportgen.dom.DocumentPart` | `unzipTemplate` | `zipTemplate` | `fill`

### More About

- “Create a Report Program” on page 13-3
- “Templates for DOM API Report Programs” on page 13-22
- “Form-Based Reporting” on page 13-24

## Report Formatting

The DOM API supports, but does not require, use of templates to generate reports. As this example illustrates, you can use the API to create scripts that generate and format content without the use of templates. The price you pay is a script that is less efficient and maintainable than it would be if it uses templates.

### Generate the Population Report

Click any of the following to generate the population report.

- `rptpop('html')`
- `rptpop('pdf')`
- `rptpop('docx')`

### Edit the RPTPOP Function

1. Click `edit('rptpop.m')` to edit the `rptpop` function.
2. Click any of the following

- `rptpop('html')`
- `rptpop('pdf')`
- `rptpop('docx')`

to run the edited report.

### See Also

`mlreportgen.dom.Document` | `mlreportgen.dom.Heading` | `mlreportgen.dom.Border` | `mlreportgen.dom.Image` | `mlreportgen.dom.Paragraph`

### More About

- “Report Formatting Approaches” on page 13-17

## Create and Use a Custom Finder

The MATLAB Report Generator report generation API supports creation of finders that search data containers for specified objects and return the results in reportable form. Finders allow you to separate search logic from report logic in your report generators. Finders also promote reuse of search logic, thereby speeding development of report generators. This example shows how to develop and use a finder to generate a report.

### Define a Finder

Creating a finder entails creating a MATLAB class that defines the finder's properties and behavior. The following sections explain the steps needed to create a finder class. The explanation uses a class named `GrantFinder` as an example. The code for the class resides in a file, `GrantFinder.m`, that accompanies this script. The `GrantFinder` class defines a finder capable of finding and formatting grants awarded by the U.S. National Endowment for the Humanities (NEH).

### Create a Skeleton Class Definition File

Use the **MATLAB Editor** (not the **Live Editor**) to create a skeleton class definition for your finder, for example

```
]classdef GrantFinder
-end
```

### Specify Finder Base Class

Specify the Report API's `mlreportgen.finder.Finder` class as the base class for your finder.

```
classdef GrantFinder < mlreportgen.finder.Finder
end
```

This base class defines properties that are common to finders, including

- **Container:** a property used to reference the container to be searched by the finder. For example, the `GrantFinder` use this property to store a reference to a grant database that it creates.
- **Properties:** a property used by finder clients to specify property values that an object must have to satisfy a search. For example, this property allows a `GrantFinder` client to specify the grant properties that a grant must have to be returned as a result of a search of the NEH grant database.

The `mlreportgen.finder.Finder` class specifies other properties and methods that your finder class definition must define. This ensures that your finder works with the Report API.

### Define a Finder Constructor

Define a function that creates an instance of your finder, for example,



```

function this = GrantFinder()
    %GrantFinder Create an instance of a grant finder

    % Assume that the grant XML file is in the current directory.
    this@mlreportgen.finder.Finder(xmlread('NEH_Grants2010s.xml'))

    % Initialize the finder
    reset(this)

end

```

The `GrantFinder` constructor uses MATLAB's `xmlread` function to read the grant XML file from disk and convert it to a Java DOM document. It then passes the DOM document to the `mlreportgen.finder.Finder` constructor, which sets the Java DOM document as the value of the finder's `Container` property. Storing the NEH database as a Java DOM document allows the finder to use third-party Java software to search the data base. See Xerces Java DOM API for more information.

The constructor also calls a `reset` function that initializes variables used to search the grant database. The `GrantFinder` class defines this function. Similarly, your class must define a `reset` function. The `reset` function ensures that a client can use your finder to conduct multiple searches of its container. See **Define a Reset Function** for more information.

### Define a find Method

Define a method to search the finder container for objects that meet user-specified constraints. The `find` method must return an array of result objects that contain the objects that it finds. Result objects are objects of base type `mlreportgen.finder.Result`. Returning the find results as result objects allows a user of your finder to add the results to a report or report chapter. See **Define a Finder Result** for more information. Returning the find results as a MATLAB array allows you to use for loop to process search results, for example

```

import mlreportgen.report.*
r = MyReport('myreport');
f = MyFinder(container);
] for result = find(f)
    add(r, result);
end
close(r);

```

The `GrantFinder`'s `find` method illustrates definition of a `find` method.

```

function results = find(this)
    %find Search the grant database and return the results
    % results = find(grantFinder) searches the NEH grant
    % database that resides in grantFinder, using the optional
    % constraints specified by grantFinder's Properties
    % property. This method returns the results as an array of
    % result objects of type GrantResult.
    %
    % See also GrantFinder.Properties, GrantResult
    results = [];

    % Get a list of the XML objects that represent the grants
    % that meet this finder's search criteria.
    getNodeList(this);

    % Convert the XML objects to an array of GrantResult nodes.
    % Return the results.
    for i = 1:this.NodeCount
        node = this.NodeList.item(i-1);
        results = [results GrantResult(node)]; %#ok<AGROW>
    end
end
end

```

This `find` method uses `getNodeList`, a search utility function that the `GrantFinder` class defines (see **Define a Search Utility Method**) to search the grant database for grants that meet property value constraints specified by the finder's `Properties` property. The `getNodeList` function sets an internal property named `NodeList` to the result of its search. The result is a Java DOM `NodeList` object (`NodeList`) that contains the search results as a list of Java DOM elements (`Element`).

The `find` method then converts this node list to an array of result objects of type `GrantResult`. It uses the `GrantResult` constructor to create a grant result object from the Java DOM `Element` object that contains the grant data.

### Define `hasNext` and `next` Methods

Your finder class definition must define `hasNext` and `next` methods. On its first invocation, your `hasNext` method must create a queue of search results and return `true` if the queue is not empty. On subsequent invocations the `hasNext` method must return `true` if the queue is empty, `false` otherwise. Your `next` method must return the first result in the queue on its first invocation, the next result, on its next invocation, and so on, until the queue is empty.

These methods are intended to allow a client of your finder to use a MATLAB `while` loop to search your finder's container, for example,

```

import mlreportgen.report.*
r = MyReport('myreport');
f = MyFinder(container);
]while hasNext(f)
    add(r, next(f));
-end
close(r);
|

```

The GrantFinder class illustrates a hasNext method.

```

function tf = hasNext(this)
    %hasNext Return true if the grant queue is not empty
    % tf = hasNext(grantFinder) creates a search result queue
    % the first time it is called. The queue contains grants that
    % reside in grantFinder and that match the search criteria
    % specified by grantFinder's Properties property. This method
    % returns true if the grant queue is not empty.
    %
    % See also GrantFinder, GrantFinder.next, GrantResult

    if this.IsIterating
        if this.NextNodeIndex <= this.NodeCount
            tf = true;
        else
            tf = false;
        end
    else
        getNodeList(this);
        if this.NodeCount > 0
            this.NextNodeIndex = 1;
            this.IsIterating = true;
            tf = true;
        else
            tf = false;
        end
    end
end
end

```

This method first checks whether it has already created a search queue as indicated by the finder's `IsIterating` property. If the queue already exists and is not empty, this method returns `true`. If the queue exists and is empty, this method returns `false`. If the queue does not yet exist (i.e., this is the method's first invocation), the `hasNext` method creates a result queue as follows. First, it uses its internal `getNodeList` method to get the grants that meet the search criteria specified by the finder's `Properties` property. The `getNodeList` method sets an internal finder property named `NodeCount` to the number of results found. If `NodeCount` is greater than zero, the `hasNext` method sets an internal property named `NextNodeIndex` to 1. The finder's `next` method uses this property to save

the state of the search queue, that is the next item in the queue. Finally, if the queue is not initially empty, the finder returns `true`; otherwise, `false`.

The `GrantFinder`'s `next` method operates on the queue created by the `hasNext` method.

```

function result = next(this)
    %next Returns the next item in the grant queue
    % result = next(grantFinder) returns the first item in the
    % grant queue the first time it is invoked. It returns the
    % next item in the queue on subsequent invocations. The
    % next result is an object of GrantResult type.
    %
    % Note: invoke hasNext to create the grant queue.
    %
    % See GrantFinder, GrantFinder.hasNext, GrantResult

    % this.IsIterating is set by hasNext to indicate that
    % it has created a search queue.
    if this.IsIterating
        % this.NextNodeIndex and this.NodeCount are initialized
        % by hasNext.
        if this.NextNodeIndex <= this.NodeCount

            % Convert the next grant XML element to a GrantResult
            % object
            result = GrantResult(...
                this.NodeList.item(this.NextNodeIndex-1));

            % Update the next node index
            this.NextNodeIndex = this.NextNodeIndex+1;

        else
            % This condition can occur if the client invokes
            % next without first invoking hasNext to determine
            % whether any more grants exist.
            error('No more grants exist')
        end
    else
        % This condition can occur if a client invokes this method
        % without first invoking hasNext to create the search
        % result queue.

        % Reset the queue variables.
        reset(this);

        % Initialize the queue.
        if hasNext(this)
            % Returns the first item in the queue.
            result = next(this);
        else
            % This condition can occur if no grants meet the
            % client's search criteria.
            error('No grants exist')
        end
    end
end
end
end

```

**Define a Search Utility Method**

Your finder's `find` and `hasNext` methods must search your finder's container for objects that satisfy search constraints. You should consider defining a search utility that both methods can use. For example, the `GrantFinder` `hasNext` and `next` methods both delegate searching to an internal utility named `getNodeList`. The `getNodeList` method in turn delegates searching to an XML document search API named `XPath` (see [XPath Tutorial](#)).

```

function getNodeList(this)
    %getNodeList Get grant elements that match search criteria
    % This is an internal method used to find grant elements. It
    % sets this finder's NodeList property to the results of the
    % search.

    if isempty(this.Properties)
        % No search criteria. Set this.NodeList to all grants in
        % the data base.
        this.NodeList = this.Container.getElementsByTagName('Grant');
    else
        % Use Java DOM's XPath API to find grants that
        % match the search criteria specified by this finder's
        % Properties property. For information on this API, see
        % https://www.ibm.com/developerworks/xml/tutorials/x-xpath/x-
        % Create an XPath expression of the form
        %
        % '/Grants/Grant[matches(p1,v1) and matches(p2,v2) ...]'
        %
        % where p1,v1,p2,v2... are the property-value pairs
        % specified by this finder's Properties property.
        %
        % The XPath expression finds all Grant elements that
        % match the specified properties. Use of the XPath matches
        % function allows the property values to be regular
        % expressions, permitting complex searches, e.g., searching
        % for grants that satisfy a range of values for a property.

        % Import the XPath API classes.
        import javax.xml.xpath.*

        %Create the XPath expression
        factory = GrantFinder.getXPathFactory;
        xPath = factory.newXPath;
        expr = xPath.compile(sprintf('/Grants/Grant[%s]', ...
            makePredicate(this)));

        this.NodeList = expr.evaluate(this.Container, ...
            XPathConstants.NODESET);
    end
    this.NodeCount = this.NodeList.getLength();
end

```

### Create an InvalidPropertyNames Property

Your finder must define a property named `InvalidPropertyNames` that specifies object properties that cannot be used to constrain a search. The `mlreportgen.finder.Finder` base class uses this property to verify that user-specified search properties specified by your finder's `Properties` property are valid. If not, the base class throws an error. In other words, if a client sets your finder's `Properties` property to invalid properties, the base class throws an error. In this way, the Report API's base finder handles property validity checking for your finder.

If your finder can use any search object property as a search constraint, it should set the `InvalidPropertyNames` property empty. For example, the `GrantFinder` can handle any grant property. It therefore sets this property empty:

```
properties (Constant, Hidden)
    InvalidPropertyNames = {}
end
```

### Define a reset Method

A finder must be able to support multiple searches to avoid the need to create a finder for every search. For this reason, the Report API's base finder class forces your finder class to define a `reset` method that resets variables used by your finder's search logic, for example,

```
function reset(this)
    %reset Resets the finder's search queue
    % reset(grantFinder) initializes the grantFinder's search
    % queue variables.
    %
    % Note: this method is declared in mlreportgen.find.Finder.
    % Subclasses must define it.

    this.NodeList = [];
    this.IsIterating = false;
    this.NodeCount = 0;
    this.NextNodeIndex = 0;
end
```

### Define a Finder Result

If a suitable definition does not exist, you must create a class to define the result objects returned by your finder. This section shows how to define a finder result object. It uses a class named `GrantResult` as an example. The `GrantResult` class defines results returned by the `GrantFinder` class used as an example in the **Define a Finder** section. The `GrantResult.m` file that accompanies this script contains the code for the `GrantResult` class. Defining a finder result entails the following tasks.

#### Specify the Result Base Class

Define `mlreportgen.finder.Result` as the base class for your result class, for example,



```

classdef GrantResult < mlreportgen.finder.Result
    %GrantResult Result of an NEH grant search
    % This class is used by GrantFinder objects to return the results
    % of searching an NEH grant database.
    %
end

```

### Define Object Property

Define a property named `Object` that clients of your result object can use to access the found object that your result object contains. Specify `protected` as the `SetAccess` value of your finder's `Object` property. This ensures that only your result can specify the found object that it contains.

```

properties (SetAccess = protected)

    %Object Java grant object
    % The value of this object is the Java object that represents
    % the grant this result object reports. You can use the Java
    % object to get grant data not available through this result
    % object's other properties.
    Object
end

```

Your result constructor must set the found object as the value of its `Object` property. Your result constructor can use the base class constructor to perform this task, for example,

```

function this = GrantResult(grant)
    %GrantResult Construct a grant object
    % result = GrantResult(grantObj) creates a result object from
    % grant, a Java DOM object containing a grant's properties.
    % The grant argument must be of type org.w3c.dom.Element.
    this@mlreportgen.finder.Result(grant);
end

```

### Expose Found Object Properties

Your result's `Object` property allows a client to access the found object and therefore its properties. However, accessing the properties can require extra code or specialized knowledge. You may want to expose some or all of the found object's properties as properties of the result object. For example, the `GrantResult` class exposes the following subset of a grant's properties.

`properties`

```
%Title Title of this grant
% Value is the content of the Java grant's ProjectTitle element
Title char = ''

%Institution Institution awarded this grant
% Value is the content of the Java grant's Institution element
Institution char = ''

%Location Location of institution awarded grant
% Value is a character array formed from the Java element's
% InstCity, InstState, InstPostalCode, and InstCountry elements,
% for example, 'West Barnstable, MA 02668-1599 USA'.
Location char = ''

%AwardAmount Amount of grant
% Value is the content of the Java grant's AwardAmount element
AwardAmount char = ''

%YearAwarded Year grant was awarded
% Value is the content of the Java grant's YearAwarded element
YearAwarded char = ''

%ToSupport Purpose of grant
% Value is the content of the Java grant's ToSupport element
ToSupport char = ''

%Participant Name and title of grant manager
% Value is a string array formed from the content of the Java
% grant's Grant/Participant/Firstname, Lastname, and
% ParticipantTypeID elements, for example,
%
% 'John Doe, Project Director'
Participant char = ''

%Tag User-defined result property
Tag
end
```

---

This saves the client of the grant finder result object from having to extract these properties itself. Your result's constructor should extract the values of the properties to be exposed and set the corresponding result properties to the extracted values, for example,

```

function this = GrantResult(grant)
    %GrantResult Construct a grant object
    % result = GrantResult(grantObj) creates a result object from
    % grant, a Java DOM object containing a grant's properties.
    % The grant argument must be of type org.w3c.dom.Element.
    this@mlreportgen.finder.Result(grant);

    this.Title = getGrantProperty(this, 'ProjectTitle');
    this.Institution = getGrantProperty(this, 'Institution');
    this.Location = sprintf('%s, %s %s %s', ...
        getGrantProperty(this, 'InstCity'), ...
        getGrantProperty(this, 'InstState'), ...
        getGrantProperty(this, 'InstPostalCode'), ...
        getGrantProperty(this, 'InstCountry') ...
    );
    this.AwardAmount = getGrantProperty(this, 'AwardOutright');
    this.YearAwarded = getGrantProperty(this, 'YearAwarded');
    this.ToSupport = getGrantProperty(this, 'ToSupport');
    this.Participant = sprintf('%s %s, %s', ...
        getParticipantProperty(this, 'Firstname'), ...
        getParticipantProperty(this, 'Lastname'), ...
        getParticipantProperty(this, 'ParticipantTypeID') ...
    );

end

```

Note that `GrantResult` combines some of the grant properties into a single exposed property. For example, it exposes a grant's `InstCity`, `InstState`, `InstPostalCode`, and `InstCountry` properties into a single result property named `Location`.

In this example, the constructor uses internal methods to extract the grant properties from the grant object, which is a Java DOM `Element` object, for example,

```

function propValue = getGrantProperty(this, propName)
    %getGrantProperty Get the value of a grant property
    %   propValue = getGrantProperty(thisResult, propName)
    %   returns the value of the grant property specified by
    %   propName as a string. |

    % Use the Java DOM getElementByTagName method to get the
    % element propName from this.Object, whose value is a Java
    % DOM element containing the grant data.
    nl = this.Object.getElementsByTagName(propName);

    % Assume that the grant element contains only one element named
    % propName.
    elem = nl.item(0);

    % Use the Java DOM getTextContent method to get the value
    % of the propName element, which is the value of the grant's
    % propName property. The getTextContent method returns a
    % Java String object. Convert and return the text content as
    % a MATLAB character array.
    propValue = char(elem.getTextContent);

end

```

### Define a getReporter Method

You must define a `getReporter` method for your result object that returns a reporter object that reports on the found object that the result object contains. This method allows a client of your finder to report on a result of a find operation simply by adding the result to a Report, Section, or Chapter object. For example,

```

import mlreportgen.report.*
r = MyReport('myreport');
f = MyFinder(container);
for result = find(f)
    add(r, result);
end
close(r);

```

A report or chapter's `add` method knows that a result object must have a `getReporter` method that returns a reporter that formats the data the result contains. So if you add a result object to a report or chapter, the `add` method invokes the result's `getReporter` method to get the result reporter and adds the result reporter to the report or reporter, causing the result data to be formatted and included in the report.

The `GrantResult` class definition defines a `getReporter` method that returns a customized version of the Report API's `mlreportgen.report.BaseTable` reporter. The `BaseTable` reporter generates

a table with a numbered title. The `GrantResult` class customizes the `BaseTable` reporter to generate a table of grant properties, for example,

#### **Grant 24. Yolo County Archives Preservation Assessment**

<b>Institution</b>	County of Yolo
<b>Location</b>	Woodland, CA 95695-3448 USA
<b>Year Awarded</b>	2010
<b>Award Amount</b>	6000.0000
<b>To Support</b>	Funding supports a preservation needs assessment of the county's archival records, along with the purchase of storage furniture and preservation supplies. Materials include records of county governance, land ownership, judicial transactions, mining activity, and agricultural development in the Sacramento Valley area since the 1850s.

The following code shows how the `GrantResult` class customizes the `BaseReporter` to generate a numbered grant properties table:

```
function reporter = getReporter(this)
    %getReporter Get a reporter for this grant
    % reporter = getReporter(thisGrantResult) returns a
    % reporter of type mlreportgen.report.BaseTable that reports
    % on selected properties of the grant that this result
    % contains.
    %
    % Note: the Report API's add method invokes the getReporter
    % method of result objects and adds the resulting reporter
    % to a report or a chapter. This allows you to add this
    % include a grant in a report by adding grant results to
    % chapter or report objects.
    %
    % If you want to customize the appearance of a grant in the
    % report, you can call this method to get the reporter,
    % customize the reporter, and then add the reporter to a
    % chapter or report.

    % Import the Report API and DOM API classes. We import the
    % DOM API classes because we want to customize the title of
    % the table used to report a grant result.
    import mlreportgen.report.*
    import mlreportgen.dom.*

    % Create an instance of a BaseTable reporter.
    reporter = BaseTable;

    % Get the reporter used to create the table's title.
    titleReporter = reporter.getTitleReporter;

    % The title of a base table consists of a prefix, followed
    % by a sequence number, followed by title text. By default,
    % the prefix is 'Table'. Let's change the prefix to 'Grant'.
    titleReporter.NumberPrefix = 'Grant ';

    % Set the table title text to the title of this grant.
    titleReporter.Content = this.Title;

    % Set the base table title to the title reporter. This causes
    % the table reporter to use the title reporter to generate our
    % customized table title.
    reporter.Title = titleReporter;
```

```
% Create a cell array containing the grant data we want to
% report. We will use this cell array to create a DOM table
% containing the grant data.
info = {
    'Institution', this.Institution; ...
    'Location', this.Location; ...
    'Year Awarded', this.YearAwarded; ...
    'Award Amount', this.AwardAmount; ...
    'To Support', this.ToSupport; ...
};

if hasParticipant(this)
    info = [info; {'Participant', this.Participant}];
end

% Create a DOM table containing the grant data.
table = Table(info);

% The first column of our table contains the names of the
% grant properties being reported. Let's make this column
% wide enough to accommodate the longest property name. Let's
% also use a bold font to render the property names, thereby
% visually distinguishing them from the property values in the
% adjacent column.

% Use the DOM API's column spec group and column spec objects
% to specify the format of the property name column. Create
% a group object to contain the specs for the individual
% columns.
grps(1) = TableColSpecGroup;

% Create a column spec object to specify the format of the
% first, i.e., property name, column of the table.
spec = TableColSpec;

% Specify the width and font formats of the first column.
spec.Style = {Width('1.3in'), Bold};

% Create a col specs array and add the first column spec to
% the spec array.
specs(1) = spec;

% Create an object to specify the width of the second, i.e.,
% property value, column of the table.
spec = TableColSpec;

% Specify the width of the column.
spec.Style = {Width('4.5in')};
```

```
% Add the col spec to the col spec array.
specs(2) = spec;

% Create a col spec group array and add the col specs array to
% the groups array.
grps(1).ColSpecs = specs;

% Add the groups array to the table.
table.ColSpecGroups = grps;

% Set the grant properties table to be the content of the
% table reporter. This causes the reporter to include the
% group properties table along with our customized table title
% in a report when the reporter is added to the report.
reporter.Content = table;
end
```

### Use a Finder

This script shows how to use a finder to generate a report. This script uses the GrantFinder example used in the Define a Finder section to generate a PDF report on NEH grants to institutions in selected states from 2010 through 2012. The script performs the following tasks.

### Import the Report Generator API

Import the classes included in the MATLAB Report Generator's Report API. Importing the classes allows the script to use unqualified (i.e., abbreviated) names to refer to the classes.

```
import mlreportgen.report.*
import mlreportgen.dom.*
```

### Create a Report Container

Create a PDF container for the report, using the Report API's `mlreportgen.report.Report` class. Note that because the script imports the Report API, it can refer to the class by its unqualified name.

```
rpt = Report('grant', 'pdf');
```

### Create the Report Title Page

Add a title page to the report, using the Report API's `TitlePage` class.

```
add(rpt, TitlePage( ...
    'Title', 'NEH Grants', ...
    'Subtitle', 'By State from 2010-2012', ...
    'Image', 'neh_logo.jpg', ...
    'Author', 'John Doe' ...
));
```

### Create the Report Table of Contents

Add a table of contents, using the Report API's `TableOfContents` class.

```
add(rpt, TableOfContents);
```



## Find the Report Data

Use an array of structures to specify the states to be included in this report. Each structure contains the data for a specific state:

- **Name:** name of the state
- **PostalCode:** the state's postal (zip) code
- **Grants:** grants made to institutions in the state. This field is initially empty.
- **NGrants:** number of grants made to institutions in this state (initially empty)

```
states = struct( ...
    'Name', {'California', 'Massachusetts', 'New York'}, ...
    'PostalCode', {'CA', 'MA', 'NY'}, ...
    'Grants', cell(1,3), ...
    'NGrants', cell(1,3) ...
);
```

Use a grant finder to populate the Grants and NGrants fields of the state structures. Create the grant finder.

```
f = GrantFinder;
```

Loop through the state array. For each state, use the finder's Properties property to constrain the search for grants awarded to the state. Use these grant properties to constrain the search:

- **InstState:** Specifies the postal code of the state in which the institution that received the grant is located.
- **YearAwarded:** Specifies the year in which the grant was awarded.

The finder treats property values as regular expressions. Exploit this fact to specify a range of values, 2010-2012, as the value of the YearAwarded property.

```
n = numel(states);
for i = 1:n
    f.Properties = [
        {'InstState', states(i).PostalCode}, ...
        {'YearAwarded', '201[0-2]'}];
    states(i).Grants = find(f);
    states(i).NGrants = numel(states(i).Grants);
end
```

## Create the Grant Summary Chapter

Create a grant summary as the first chapter of the report. The grant summary chapter contains a title and a grant summary table. Each row of the table lists the total number of grants and total amount of money awarded to institutions in the state for the years 2010-2012. States appear in the table in descending order of number of grants. Each state is hyperlinked to the chapter that details the grants awarded to it.

## Chapter 1. Grant Summary

State	Grants Awarded	Amount Awarded
<a href="#">New York</a>	308	\$31,711,344.61
<a href="#">California</a>	235	\$27,413,312.75
<a href="#">Massachusetts</a>	184	\$22,789,551.14

### Create the Summary Chapter Container

Start by creating a chapter container. .

```
ch = Chapter('Title', 'Grant Summary');
```

### Create the Contents of the Grant Summary Table

Start by creating a Java currency formatter. Use this object to format the dollar amount of the grants awarded to a state.

```
currencyFormatter = java.text.NumberFormat.getCurrencyInstance();
```

Create a cell array containing the contents of the table header.

```
header = {'State', 'Grants Awarded', 'Amount Awarded'};
```

Preallocate a cell array to contain the table body contents. The cell array has Rx3 rows and columns where R is the number of states and 3 is the number of summary items reported for each state.

```
body = cell(numel(states), 3);
```

Sort the states array by the number of grants awarded to them, using the MATLAB `sort` function. The `sort` function returns `ind`, an array of indices to the states array. The first index of the `ind` array is the index of the state with the most grants, the second, with the second most number of grants, etc.

```
[~, ind] = sort([states.NGrants], 'descend');
```

Loop through the states by number of grants, filling in the summary information for each state. Use a variable, `rowIdx`, as an index to the cell array row corresponding to the current state.

```
rowIdx = 0;
```

The following line rearranges the `states` array in order of grants received and creates a `for` loop that assigns each structure in the sorted `states` array to the variable `state` on each iteration of the loop.

```
for state = states(ind)
```

Update the row index to point to the cell array row corresponding to the current state.

```
    rowIdx = rowIdx+1;
```

The script enters a hyperlink to the grant details chapter for the state as the first entry in the table for the state, for example,

## California

The following line uses a DOM `InternalLink` constructor to create the hyperlink. The `InternalLink` constructor takes two arguments, a link target id and the text of the hyperlink. The script uses the current state's postal code as the link target id and the state's name as the link text. Later on, when the script creates the grant details chapter, it inserts a link target in the chapter title whose id is the state's postal code. This completes creation of the hyperlink.

```
body(rowIdx, 1) = {InternalLink(state.PostalCode, state.Name)};
```

Assign the total number of grants for this state to the second item in its cell array row.

```
body(rowIdx, 2) = {state.NGrants};
```

Compute the total amount awarded to this state.

```
totalAwarded = 0;
for grant = state.Grants
    totalAwarded = totalAwarded + str2double(grant.AwardAmount);
end
```

Use the currency formatter to format the total amount as a dollar amount, for example,

**\$27,413,312.75**

and assign the formatted result as the third and final item in the cell array for this state.

```
body(rowIdx,3) = {char(currencyFormatter.format(totalAwarded))};
end
```

To create the summary table, pass the header and body cell arrays to the constructor of a `mlreportgen.dom.FormatTable` object.

```
table = FormatTable(header, body);
```

A formal table is a table that has a header and a body. The `FormatTable` constructor takes two arguments: a cell array that specifies the contents of the table's header and a cell array that specifies the contents of its body. The constructor converts the cell array contents to DOM `TableRow` and `TableEntry` objects that define the table, saving the script from having to create the necessary table objects itself.

### **Format the Grant Summary Table**

At this point, the summary table looks like this:

State	Grants	AwardedAmount	Awarded
<a href="#">New York</a>	308		\$31,711,344.61
<a href="#">California</a>	235		\$27,413,312.75
<a href="#">Massachusetts</a>	184		\$22,789,551.14

This is not very readable. The heading has the same format as the body and the columns are not spaced apart.

In the following steps, the script adjusts the header text formatting to look like this:

State	Grants Awarded	Amount Awarded
<a href="#">New York</a>	308	\$31,711,344.61
<a href="#">California</a>	235	\$27,413,312.75
<a href="#">Massachusetts</a>	184	\$22,789,551.14

First the script specifies the width and alignment of the table columns, using a DOM `TableColSpecGroup` object. A `TableColSpecGroup` object specifies the format of a group of columns. The summary table has only one group of columns so the script needs to create only one `TableColSpecGroup` object.

```
grp = TableColSpecGroup;
```

The `TableColSpecGroup` object lets the script specify the default style of the table's columns. The script specifies 1.5in as the default width of the columns and center alignment as the default column alignment.

```
grp.Style = {HAlign('center'), Width('1.5in')};
```

The script uses a `TableColSpec` object to override the default column alignment for the first column.

```
specs(1) = TableColSpec;
specs(1).Style = {HAlign('left')};
grp.ColSpecs = specs;
table.ColSpecGroups = grp;
```

**Note** The script could use as many as three `TableColSpec` objects, one for each table column, to override the group column styles. The first `TableColSpec` object applies to the first column, the second to the second column, etc. The script needs to assign only one column spec object to the group because it is overriding the default style only for the first column. However, if it needed to change only the third column, it would have to assign three column spec objects, leaving the `Style` property of the first two column spec objects empty.

The default table style crowds the table entries. So the script uses a DOM `InnerMargin` format object to create some space above the entries to separate them from the entries in the row above them. An `InnerMargin` object creates space (inner margin) between a document object and the object that contains it, for example, between the text in a table entry and the table entry's borders. The `InnerMargin` constructor optionally takes four arguments, the left, right, top, bottom inner margins of a document object.

The script use this constructor to create a top inner margin format of 3 points. It then assigns this format to the style of the entries in the summary table's body section.

```
table.Body.TableEntriesStyle = {InnerMargin('0pt', '0pt', '3pt', '0pt')};
```

Finally the script format the table header to consist of bold, white text on a gray background:

```
table.Header.row(1).Style = {Bold, Color('white'), BackgroundColor('gray')};
```

State	Grants Awarded	Amount Awarded
-------	----------------	----------------

**Add the Summary Chapter to the Report**

```
add(ch, table);
add(rpt, ch);
```

**Create the Grant Details Chapters**

Loop through the state structures.

```
for state = states
```

For each state create a chapter to hold the state's grant details. Insert a link target into the chapter title to serve as a target for the hyperlink in the summary table in the first chapter.

```
    ch = Chapter('Title', {LinkTarget(state.PostalCode), state.Name});
```

Loop through the grant results for the state.

```
    for grant = state.Grants
```

For each grant result, add the result to the chapter.

```
        add(ch, grant);
```

A grant result has a `getReporter` method that returns a reporter that creates a table of selected grant properties. The chapter `add` method is preconfigured to get a result's reporter and add it the chapter. Thus, adding a grant to a chapter is tantamount to adding the result property table to the chapter, for example,

**Grant 24. Yolo County Archives Preservation Assessment**

<b>Institution</b>	County of Yolo
<b>Location</b>	Woodland, CA 95695-3448 USA
<b>Year Awarded</b>	2010
<b>Award Amount</b>	6000.0000
<b>To Support</b>	Funding supports a preservation needs assessment of the county's archival records, along with the purchase of storage furniture and preservation supplies. Materials include records of county governance, land ownership, judicial transactions, mining activity, and agricultural development in the Sacramento Valley area since the 1850s.

```
        end
    add(rpt, ch);
end
```

**Close the Report Object**

Closing the report object generates the PDF output file (`grant.pdf`) that the report object specifies.

```
close(rpt);
```

**Display the report**

```
rptview(rpt);
```

## Appendix: The NEH Grant Database

The source of the database used in this example is the National Endowment for the Humanities (NEH). The database contains information on NEH grants for the period 2010-2019. It contains about 6000 records in XML format. It is available at NEH Grant Data. This example uses a local copy of the database XML file, `NEH_Grants2010s.xml`.

The database consists of a `Grants` element that contains a set of `Grant` elements each of which contains a set of grant data elements. The following is an extract from the database that illustrates its structure:

```
<Grant AppNumber="TD-50102-10">
  <ApplicantType>O</ApplicantType>
  <Institution>Center for Independent Documentary, Inc.</Institution>
  <OrganizationType>Four-Year College</OrganizationType>
  <InstCity>Boston</InstCity>
  <InstState>MA</InstState>
  <InstPostalCode>02135-1032</InstPostalCode>
  <InstCountry>USA</InstCountry>
  <CongressionalDistrict>7</CongressionalDistrict>
  <Latitude>42.36309</Latitude>
  <Longitude>-71.14034</Longitude>
  <CouncilDate>2009-07-01</CouncilDate>
  <YearAwarded>2010</YearAwarded>
  <ProjectTitle>Mysticism and Monotheism</ProjectTitle>
  <Program>America's Media Makers: Development Grants</Program>
  <Division>Public Programs</Division>
  <ApprovedOutright>65000.0000</ApprovedOutright>
  <ApprovedMatching>0.0000</ApprovedMatching>
  <AwardOutright>65000.0000</AwardOutright>
  <AwardMatching>0.0000</AwardMatching>
  <OriginalAmount>65000.0000</OriginalAmount>
  <SupplementAmount>0.0000</SupplementAmount>
  <BeginGrant>2010-02-01</BeginGrant>
  <EndGrant>2010-08-31</EndGrant>
  <ProjectDesc>None</ProjectDesc>
  <ToSupport>Scripting for a two-hour television broadcast and companion website on mystical practice and its contemporary role
    in Judaism, Christianity, and Islam.</ToSupport>
  <PrimaryDiscipline>Comparative Religion</PrimaryDiscipline>
  <SupplementCount>0</SupplementCount>
  <ParticipantCount>1</ParticipantCount>
  <Participant>
    <Firstname>Christine</Firstname>
    <Lastname>Herbes-Sommers</Lastname>
    <Institution>Center for Independent Documentary, Inc.</Institution>
    <City>Boston</City>
    <State>MA</State>
    <Country>USA</Country>
    <ParticipantTypeID>Project Director</ParticipantTypeID>
    <ValidFr>2009-02-03</ValidFr>
  </Participant>
  <DisciplineCount>1</DisciplineCount>
  <Discipline>
    <Name>Comparative Religion</Name>
  </Discipline>
</Grant>
```

## See Also

`mlreportgen.finder.Finder` | `mlreportgen.report.Report` |  
`mlreportgen.report.Chapter` | `rptview`

## **More About**

- “Create a Report Program” on page 13-3

## Resolve Error from Closing a PDF Document on a Cloud Drive

### Issue

Generating a PDF report on MATLAB Drive or other cloud drive can result in an error that is caused by file contention between the report generation software and the cloud drive synchronization software. The error is similar to:

```
Error using mlreportgen.report.internal.Document/closeError closing document
package: No content type was found for part:/.MATLABDriveTagError in
mlreportgen.report.ReportBase/close
```

### Possible Solutions

Generate PDF reports on a local drive that does not synchronize with the cloud. Consider writing a script that generates a report on a local drive and then copies the report to the cloud drive.

### See Also

`mlreportgen.dom.Document` | `mlreportgen.report.Report` | `slreportgen.report.Report`



# Programmatic PowerPoint Presentation Creation

---

- “Create a Presentation Generator” on page 14-2
- “Create PPT Objects” on page 14-7
- “Import the PPT API Package” on page 14-9
- “Get and Set PPT Object Properties” on page 14-10
- “Create a Presentation Object to Hold Content” on page 14-12
- “Generate a Presentation” on page 14-13
- “Display Presentation Generation Messages” on page 14-14
- “Create a Standalone Application from a Presentation Program” on page 14-17
- “Presentation Formatting Approaches” on page 14-18
- “Presentation Format Inheritance” on page 14-21
- “Set Up a PowerPoint Presentation Template” on page 14-23
- “Access PowerPoint Template Elements” on page 14-32
- “Define a Style Using Format Objects” on page 14-37
- “Use Format Properties” on page 14-39
- “Update Presentation Content Programmatically” on page 14-41
- “Create a Presentation Programmatically” on page 14-49
- “Add Slides” on page 14-56
- “Add and Replace Presentation Content” on page 14-58
- “Create and Format Text” on page 14-65
- “Create and Format Paragraphs” on page 14-67
- “Create and Format Tables” on page 14-69
- “Create and Format Pictures” on page 14-76
- “Create and Format Links” on page 14-78
- “Generate a Presentation From the Results of a MATLAB Application” on page 14-84

## Create a Presentation Generator

### In this section...

“Update Presentation Content” on page 14-3

“Two Ways to Use the PPT API” on page 14-5

“PPT API Applications and PowerPoint Templates” on page 14-5

“Template Elements” on page 14-5

You can use the MATLAB API for PowerPoint (PPT API) to update and create PowerPoint presentations programmatically. For example, this MATLAB script creates a presentation that has a title page and one content slide with a bulleted list.

```
import mlreportgen.ppt.*;

ppt = Presentation("mySlides.pptx");
open(ppt);
slide1 = add(ppt,"Title Slide");
replace(slide1,"Title",'My Presentation');
replace(slide1,"Subtitle","Create a Presentation Program");

slide2 = add(ppt,"Title and Content");
para = Paragraph("First Content Slide");
para.FontColor = "blue";
replace(slide2,"Title",para);

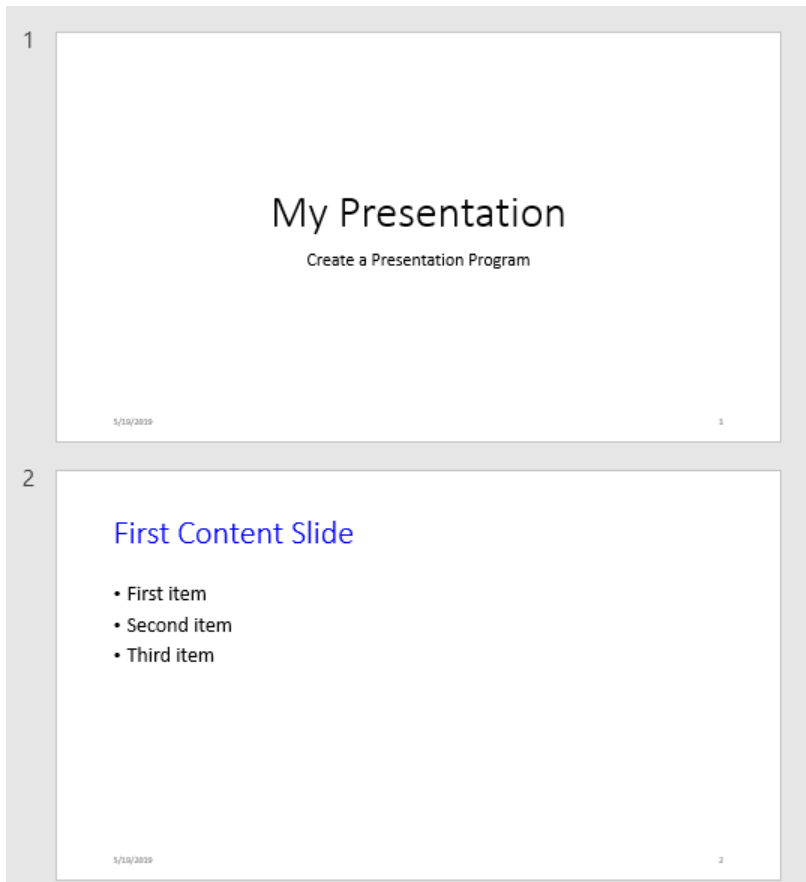
replace(slide2,"Content",["First item","Second item","Third item"]);

close(ppt);
```

After you create the presentation, which is named `MySlides.pptx`, you can open it.

```
rptview(ppt)
```

The generated presentation `MySlides.pptx` includes these two slides.



## Update Presentation Content

PPT API programs generally include code that:

- Imports the `mlreportgen.ppt` API package. To omit the package name when you invoke PPT API object constructors and method, import the package.

```
import mlreportgen.ppt.*;
```

- Creates a Presentation object to:
  - Hold the presentation contents
  - Specify the output location for the generated presentation
  - Indicate the PowerPoint template

The following code creates a presentation using the template from the presentation in the file `mySlides.pptx` and overwrites `mySlides.pptx` with the new presentation.

```
slidesFile = "mySlides.pptx";
ppt = Presentation(slidesFile, slidesFile);
open(ppt);
```

- Adds or replaces slide content.

```
slide2 = ppt.Children(2);
contents = find(slide2, "Title");
```

```

replace(contents,Paragraph("Modified Content Slide"));

contents = find(slide2,"Content");
datePara = Paragraph("Fourth item: Updated item");

add(contents,datePara);

```

The PPT API replaces PowerPoint template placeholders with content defined in the program. In the template, you can interactively add placeholders or rename placeholders for your program to interact with.

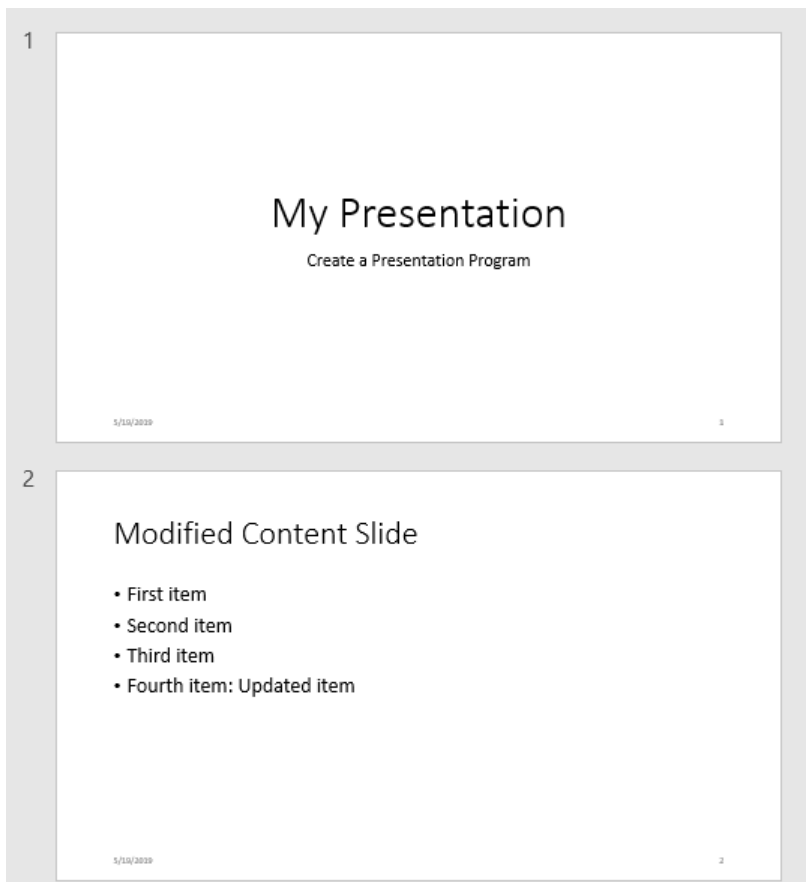
- Closes the presentation, which generates the content and formatting of the presentation.

```
close(ppt);
```

You can include code to open the presentation.

```
rptview(ppt)
```

The updated slide looks like this:



To see another example of a PPT API program, see “Generate a Presentation From the Results of a MATLAB Application” on page 14-84.

## Two Ways to Use the PPT API

You can create a PPT API program that:

- Replaces content in, or adds content to, an existing PowerPoint presentation
- Generates a complete PowerPoint presentation

### Add Content to an Existing Presentation

To add or update content to an existing presentation without manually updating the presentation each time content changes, use the PPT API. This approach is useful when you want to use most of the content and formatting in an existing presentation.

- You can use the PPT API and MATLAB functions to generate content for a presentation from MATLAB code and Simulink models.
- You can update a presentation by overwriting the presentation file or create a separate version of the presentation with a different presentation name.

### Create a Complete Presentation

To create a complete presentation when you want to use the same content using multiple PowerPoint templates, use the PPT API.

## PPT API Applications and PowerPoint Templates

The PPT API uses PowerPoint presentations as templates to generate presentations. Templates allow you to specify the fixed content and default layout and appearance of the slides in your presentations. Your MATLAB program can use the PPT API to override the default layout and format of specific slides.

The template can be an empty presentation or a presentation with slides. You can use the following as templates for a PPT API presentation:

- The default PPT API PowerPoint template
- A customized copy of the default PPT API PowerPoint template
- An existing PowerPoint presentation whose content you want to update
- A PowerPoint template that you create or update interactively in PowerPoint

See “Set Up a PowerPoint Presentation Template” on page 14-23.

## Template Elements

PowerPoint templates include several elements that the PPT API uses to generate a presentation. To customize formatting defined in a template, modify one or more of these template elements.

PowerPoint Template Element	Purpose
Slide masters	Applies the slide master formatting globally to the presentation. Specifies a layout and formats common to a set of slide layouts
Slide layouts	Specifies a variant of a slide master layout.

<b>PowerPoint Template Element</b>	<b>Purpose</b>
Table styles	Specifies the default appearance of a table. PowerPoint defines a standard set of table styles. You cannot modify these styles but you can use the PPT API to apply these styles to tables you create and override the styles for particular tables.
Placeholders	Specifies an area of a slide layout that you can replace with text, a list, picture, table, or other content. Every placeholder has a name. You can use PowerPoint interactively to assign a name to a placeholder. You can then use the name in your PPT program to replace the placeholder with content.

## See Also

### Related Examples

- “Create PPT Objects” on page 14-7
- “Create a Presentation Object to Hold Content” on page 14-12
- “Update Presentation Content Programmatically” on page 14-41
- “Create a Presentation Programmatically” on page 14-49

## Create PPT Objects

### In this section...

“PPT Objects” on page 14-7

“Use a PPT Constructor” on page 14-7

“PPT Objects Created Without Constructors” on page 14-7

### PPT Objects

The PPT API consists of a hierarchical set of data structures, known as objects, that represent a presentation and its contents. The top of the hierarchy has an object representing the presentation. The PPT API maintains a list of objects, called the presentation children, that represent the presentation contents (slides, paragraphs, tables, pictures, etc.). Each child object, in turn, maintains a list of its contents. For example, the children of a table object are its row objects, the children of a row object are its entry objects, and so on.

The PPT API contains functions (also known as methods) to create and assemble PPT objects, such as paragraphs and tables, and add the objects to slides.

The PPT API includes format objects, such as bold and font color objects, that you can use to define formatting for presentation elements.

To generate a PowerPoint presentation file, use the PPT API. You can open, view, and edit the generated presentation as you do with any other PowerPoint presentation.

### Use a PPT Constructor

The PPT API includes a set of MATLAB functions, called constructors, that you use to create PPT objects of various types.

The name of an object constructor is the name of the MATLAB class from which the PPT API creates an object. For example, the name of the constructor for a PPT paragraph object is `mlreportgen.ppt.Paragraph`. Some constructors do not require any arguments. Other constructors can take arguments that typically specify its initial content and properties. For example, this code creates a paragraph object, `p`, whose initial content is `Slide 1`.

```
p = mlreportgen.ppt.Paragraph('Slide 1');
```

A constructor returns a handle to the object it creates. Assigning the handle to a variable allows you to append content to the object or set its properties. For example, this code appends content to the paragraph object `p`.

```
append(p, '-- In the Beginning');
```

### PPT Objects Created Without Constructors

You can use some PPT API functions to create PPT objects without including a constructor in your code. For example, to create a slide, add a slide layout to a presentation without an `mlreportgen.ppt.Slide` constructor. This code uses an `add` method for the `mlreportgen.ppt.Presentation` object `slides`. The `add` method creates a `Slide` object named `slide1` based on the `Title Slide` layout in the default PPT API PowerPoint template.

```
import mlreportgen.ppt.*;
ppt = Presentation('MySlides');

slide1 = add(ppt, 'Title Slide')

slide1 =

    Slide with properties:

        Layout: 'Title Slide'
        SlideMaster: 'Office Theme'
        Name: ''
        Style: []
        Children: [1x2 mlreportgen.ppt.TextBoxPlaceholder]
        Parent: [1x1 mlreportgen.ppt.Presentation]
        Tag: 'ppt.Slide:16'
        Id: '16'
```

### See Also

#### Functions

add

#### Classes

mlreportgen.ppt.Presentation | mlreportgen.ppt.Slide

### Related Examples

- “Import the PPT API Package” on page 14-9
- “Create a Presentation Object to Hold Content” on page 14-12



## Import the PPT API Package

All PPT class names and constructor names have the prefix `m\reportgen.ppt`. To omit the prefix in your code, insert this statement at the beginning of a PPT API program.

```
import m\reportgen.ppt.*;
```

Examples that refer to PPT API objects and functions without the `m\reportgen.ppt` prefix assume that you have imported the PPT API package.

### See Also

#### Related Examples

- “Create PPT Objects” on page 14-7
- “Get and Set PPT Object Properties” on page 14-10
- “Create a Presentation Generator” on page 14-2

## Get and Set PPT Object Properties

Most PPT objects have properties that describe the object. For example, Paragraph objects have properties such as `Bold`, `FontColor`, and `Level`. You can set the value of most object properties.

To get or set the property of PPT object, use dot notation:

- Append a period to the name of a variable that references the object.
- Add the property name after the period.

For example, this code creates a paragraph containing the text `Hello World` and colors the text green.

```
p = Paragraph('Hello World')
```

```
p =
```

```
Paragraph with properties:
```

```
    Bold: []
  FontColor: []
    Italic: []
    Strike: []
  Subscript: []
  Superscript: []
  Underline: []
    Level: []
    Style: []
  Children: [1x1 mlreportgen.ppt.Text]
  Parent: []
    Tag: 'ppt.Paragraph:1534'
    Id: '1534'
```

```
p.FontColor = 'green';
```

This code displays the properties of the first child of the paragraph `p`.

```
p.Children
```

```
ans =
```

```
Text with properties:
```

```
    Content: 'Hello World'
    Bold: []
  FontColor: []
    Italic: []
    Strike: []
  Subscript: []
  Superscript: []
  Underline: []
    Style: []
  Children: []
  Parent: [1x1 mlreportgen.ppt.Paragraph]
    Tag: 'ppt.Text:1535'
    Id: '1535'
```

## **See Also**

### **Related Examples**

- “Use Format Properties” on page 14-39
- “Create PPT Objects” on page 14-7

### **More About**

- “Presentation Formatting Approaches” on page 14-18

## Create a Presentation Object to Hold Content

Every PPT API program must create an `mlreportgen.ppt.Presentation` object to hold presentation content. To create a presentation object, use the `mlreportgen.ppt.Presentation` constructor.

If you use the constructor without arguments, the PPT API creates a presentation named `Untitled.pptx` in the current folder. The presentation uses the default PPT API PowerPoint template.

You can specify the file system path of the presentation as the first argument of the constructor.

For the second argument of the constructor, you can specify a PowerPoint template to use. This `Presentation` constructor creates a presentation called `myPresentation` in the current folder, using a PowerPoint template called `CompanyTemplate.pptx`.

```
pres = Presentation('myPresentation', 'CompanyTemplate.pptx');
```

If the template you use is an existing presentation that includes content, the new presentation that the PPT API generates includes the content in that presentation. You can replace content from the template using the PPT API.

### See Also

`mlreportgen.ppt.Presentation`

### Related Examples

- “Create a Presentation Generator” on page 14-2
- “Create PPT Objects” on page 14-7
- “Generate a Presentation” on page 14-13
- “Access PowerPoint Template Elements” on page 14-32
- “Set Up a PowerPoint Presentation Template” on page 14-23

## Generate a Presentation

To generate a PowerPoint presentation from your PPT API program, use the API to close the presentation. For example, to generate a presentation whose `Presentation` object is `slides`:

```
close(slides);
```

Generating a presentation overwrites the previous version of the presentation file. Closing a presentation creates or overwrites a `.pptx` file in the path that you specify in the `Presentation` object constructor. For example, closing this presentation creates a `MyPresentation.pptx` file in the current folder:

```
import mlreportgen.ppt.*;
slides = Presentation('MyPresentation');
add(slides, 'Title and Content');
close(slides);
```

---

**Note** If the presentation (`.pptx`) file is already open in PowerPoint, interactively close the PowerPoint presentation file before you generate the presentation using the PPT API program.

---

### See Also

### Related Examples

- “Display Presentation Generation Messages” on page 14-14

## Display Presentation Generation Messages

### In this section...

“Presentation Generation Messages” on page 14-14  
 “Display PPT Default Messages” on page 14-14  
 “Create and Display a Progress Message” on page 14-15

### Presentation Generation Messages

The PPT API can display messages when you generate a PowerPoint presentation. The messages are triggered every time a presentation element is created or appended during presentation generation.

You can define additional messages to display while a presentation generates. The PPT API provides these classes for defining messages:

- `ProgressMessage`
- `DebugMessage`
- `WarningMessage`
- `ErrorMessage`

The PPT API provides additional classes for handling presentation message dispatching and display. It uses MATLAB events and listeners to dispatch messages. A message is dispatched based on event data for a specified PPT object. For an introduction to events and listeners, see “Event and Listener Concepts”.

---

**Note** When you create a message dispatcher, the PPT API keeps the dispatcher until the end of the current MATLAB session. To avoid duplicate reporting of message objects during a MATLAB session, delete message event listeners.

---

### Display PPT Default Messages

This example shows how to display the default PPT debug messages. Use a similar approach for displaying other kinds of PPT presentation messages.

- 1 Create a message dispatcher, using the `MessageDispatcher.getTheDispatcher` method. Use the same dispatcher for all messages.

```
dispatcher = MessageDispatcher.getTheDispatcher;
```

- 2 To display debug messages, use the `MessageDispatcher.Filter` property.

```
dispatcher.Filter.DebugMessagesPass = true;
```

- 3 Add a listener using the MATLAB `addlistener` function. Specify the dispatcher object, the source and event data, and a `disp` function that specifies the event data and format for the message.

```
l = addlistener(dispatcher, 'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsText));
```

- 4 Add code that deletes the listener after the code that generates the presentation.

```
delete(l);
```

This presentation displays debug messages.

```
import mlreportgen.ppt.*;

dispatcher = MessageDispatcher.getTheDispatcher;
dispatcher.Filter.DebugMessagesPass = true;

l = addlistener(dispatcher, 'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsText));

slides = Presentation('myMessagePresentation');
titleSlide = add(slides, 'Title and Content');

p = Paragraph('Hello World:');
p.Style = {Bold(true)};
t = Text(' How are you?');
t.Bold = false;
append(p,t);

add(titleSlide, 'Content', p);

close(slides);

delete(l);
```

## Create and Display a Progress Message

This example shows how to create and dispatch a progress message. You can use a similar approach for other kinds of messages, such as warnings.

- 1 Create a message dispatcher.

```
dispatcher = MessageDispatcher.getTheDispatcher;
```

- 2 Add a listener using the MATLAB `addlistener` function.

```
l = addlistener(dispatcher, 'Message', ...
    @(src, evtdata) disp(evtdata.Message.formatAsText));
```

- 3 Dispatch the message, using the `Message.dispatch` method. Specify the dispatcher object and the message to dispatch. Here the message is a debug message called `firstSlide`, and the `Presentation` object `slides` is the source of the message.

```
dispatch(dispatcher, ProgressMessage('firstSlide', slides));
```

- 4 Add code that deletes the listener after the code that generates the presentation.

```
delete(l);
```

This presentation uses this progress message.

```
import mlreportgen.ppt.*;
pre = Presentation('myPresentation.pptx');

dispatcher = MessageDispatcher.getTheDispatcher;
l = addlistener(dispatcher, 'Message', ...
```

```
        @(src, evtdata) disp(evtdata.Message.formatAsText));

dispatch(dispatcher,ProgressMessage('starting presentation',pre));
open(pre);

titleText = Text('This is a Title');
titleText.Style = {Bold};

replace(pre,'Title',titleText);

close(pre);

delete(l);
```

## See Also

### Functions

dispatch | mlreportgen.ppt.MessageDispatcher.getTheDispatcher | formatAsHTML |  
formatAsText | passesFilter

### Classes

mlreportgen.ppt.MessageDispatcher | mlreportgen.ppt.MessageEventData |  
mlreportgen.ppt.MessageFilter | mlreportgen.ppt.DebugMessage |  
mlreportgen.ppt.ProgressMessage | mlreportgen.ppt.WarningMessage |  
mlreportgen.ppt.ErrorMessage



## Create a Standalone Application from a Presentation Program

If you have the MATLAB Compiler product, you can package your MATLAB API for PowerPoint (PPT API) programs into standalone applications. Then you can share the standalone application with others who do not have a licensed copy of MATLAB.

To enable execution of standalone PPT API programs on a system that does not have a licensed copy of MATLAB, you must include `makePPTCompilable()` before the first line of PPT API code. For example:

```
makePPTCompilable();

import mlreportgen.ppt.*

ppt = Presentation("myPresentation");
open(ppt);
slide = add(ppt,"Title Slide");
replace(slide,"Title","My Title");
close(ppt);
```

---

**Note** To enable execution of standalone DOM and Report API programs, use `makeDOMCompilable()` instead of `makePPTCompilable()`.

---

To create a standalone application from a PPT API program, use the `compiler.build.standaloneApplication` function or the **Application Compiler** app. See “Create Standalone Application from MATLAB” (MATLAB Compiler).

### See Also

`compiler.build.standaloneApplication`

### More About

- “Create Standalone Application from MATLAB” (MATLAB Compiler)
- “Create a Standalone Application from a Report Program” on page 13-118

## Presentation Formatting Approaches

With the MATLAB API for PowerPoint (PPT API), you can use a PowerPoint template and PPT API format objects and properties to specify the appearance of an object. The PPT API supports four approaches for formatting elements of a presentation.

Formatting Approach	Use
Define formatting in the PowerPoint template.	<ul style="list-style-type: none"> <li>Applying formatting globally within a presentation</li> <li>Maintaining consistency across presentations</li> <li>Extending formatting options that the PPT API provides</li> </ul>
Using the PPT API, specify format objects to define a style for a presentation object.	<ul style="list-style-type: none"> <li>Formatting a specific presentation element</li> <li>Specifying multiple format options in one statement</li> <li>Specifying complicated values such as hexadecimal color values that are used repeatedly in a program</li> <li>Extending formatting options beyond the ones that format properties of an object provide</li> <li>Defining a style to use with multiple objects</li> </ul>
Using the PPT API, set format properties of a presentation object.	<ul style="list-style-type: none"> <li>Specifying one or two basic format options for a specific presentation object</li> <li>Extending formatting options beyond those options that format properties of an object provide</li> <li>Specifying one or two basic format options for a specific presentation object</li> </ul>
In the PowerPoint software, format a generated PPT API.	<ul style="list-style-type: none"> <li>Customizing a specific version of a generated presentation</li> <li>Extending formatting options beyond those options that the format objects provide</li> </ul>

### Template Formatting

Use templates for applying formatting globally:

- Across a whole presentation, for example, the background color of slides.
- To specific kinds of elements in a presentation, for example, slide titles.

Using a PowerPoint template with the PPT API involves creating and formatting template elements such as:

- Slide masters
- Slide layouts
- Placeholders
- Table styles

Using the template to define formatting offers more formatting options than the PPT API provides. Defining formatting in the template allows you to have consistent formatting in any PPT API presentation that used the template.

To format specific content in a specific slide, consider using one of the other approaches. Adding special-case formatting elements in a template can make the template overly complex.

## Format Objects

You can define PPT API format objects and use them to specify a formatting style for presentation objects. After you create a presentation object, you can define the `Style` property for that object, using a cell array of format objects. For example:

```
import mlreportgen.ppt.*

p = Paragraph('Model Highlights');
p.Style = {FontColor('red'),Bold(true)};
```

It is a best practice to set the `Style` property by concatenating the existing value of the `Style` property and the cell array of format objects that you are adding. For example:

```
import mlreportgen.ppt.*

p = Paragraph('Model Highlights');
p.Style = [p.Style {FontColor('red'),Bold(true)}];
```

This practice prevents inadvertent removal of format objects that you previously added or that the PPT API added to synchronize the `Style` property with the format properties.

For many presentation objects, using format objects provides more formatting options than the format properties of the presentation objects. Using format objects can streamline your code: you can combine multiple formatting options in one statement and apply a defined style to multiple presentation objects.

## Format Properties

Use format properties of a PPT API presentation element for basic formatting of a specific presentation object.

After you define a presentation object, you can set values for its format properties. For example:

```
import mlreportgen.ppt.*

p = Paragraph('My paragraph');
p.FontColor = 'red';
p.Font = 'Arial';
p.FontSize = '18pt';
```

The formatting applies only to the specific object. If you want to set just one option for a presentation element, using a format property is the simplest approach.

## Interactive Formatting of Slide Content

After you generate a PPT API presentation, you can use the PowerPoint software to fine-tune the formatting.

In PowerPoint, you can use all PowerPoint formatting options, including options that you cannot specify with the PPT API, such as animation. Interactive editing of slide content of the generated

presentation allows you to customize a specific version of the presentation without impacting future versions of the presentation.

If you use PowerPoint to customize a presentation generated using the PPT API, you lose those customizations when you generate the presentation again. To preserve the interactive formatting of content, save the customized version of the presentation using a different file name.

### **See Also**

#### **Related Examples**

- “Set Up a PowerPoint Presentation Template” on page 14-23
- “Define a Style Using Format Objects” on page 14-37
- “Use Format Properties” on page 14-39

#### **More About**

- “Presentation Format Inheritance” on page 14-21
- “Access PowerPoint Template Elements” on page 14-32

## Presentation Format Inheritance

The PPT API allows you to use a PowerPoint template and PPT API format objects and properties to format presentation objects. You can combine formatting approaches.

The formatting you specify in a PowerPoint template specifies the default format of presentation content.

You can use a PPT API to format a specific presentation object. You can:

- Define format objects that you can use with a presentation object `Style` property.
- Specify a value for a format property of a presentation object.

You can combine formatting with the `Style` property and formatting with format properties. For example:

```
p = Paragraph('This is a paragraph');
p.Style = {Bold(true),Underline('wavy')};
p.FontColor = 'red';
```

If you define the same formatting characteristic using each approach, the PPT API uses the specification that appears later in the code. For example, this code specifies blue as the default color for text in a paragraph:

```
p = Paragraph('This is a paragraph');
p.Style = {FontColor('red')};
p.FontColor = 'blue';
```

Several PPT API objects are hierarchical. For example:

- You can append a `Text` object to a `Paragraph` object.
- You append `TableEntry` objects to a `TableRow` object, and you can append `TableRow` objects to a `Table` object.

The formatting for a parent object applies to its child objects. However, formats specified by the child object override the parent formatting. For example:

```
import mlreportgen.ppt.*;

ppt = Presentation('myParagraphPresentation.pptx');
open(ppt);

slide1 = add(ppt,'Title and Content');

%% Use Unicode for special characters
p = Paragraph('Parent default red text: ');
p.FontColor = 'red';

t = Text('child text object blue text');
t.FontColor = 'blue';

append(p,t);
add(slide1,'Content',p);

close(ppt);
rptview(ppt);
```



## Set Up a PowerPoint Presentation Template

A PowerPoint template specifies the fixed content and default layout and appearance of the slides in a presentation. When you create a presentation by using the PPT API, you can specify a template to use. The template can be empty or have slides. Any content in the template appears in the new presentation. If you do not specify a template, the PPT API uses a default template.

The template that you specify can be:

- A customized copy of the PPT API default template.
- An existing presentation.
- A template that you set up in PowerPoint.

To customize a copy of the default template or an existing presentation that you want to use as a template, you can use the PPT API or PowerPoint. To add or customize slide layouts, use PowerPoint.

### Specify a Template

To specify a template, provide the template name as the second argument when you create an `mreportgen.ppt.Presentation` object. For example, this code specifies the template `myTemplate.pptx`.

```
import mreportgen.ppt.*
ppt = Presentation("myPresentation.pptx", "myTemplate.pptx");
```

To use the PPT API default template, omit the second argument. For example:

```
import mreportgen.ppt.*
ppt = Presentation("myPresentation.pptx");
```

### Copy the Default Template

To copy the PPT API default template, use `mreportgen.ppt.Presentation.createTemplate`. For example:

```
templatePath = mreportgen.ppt.Presentation.createTemplate("myTemplate.pptx");
```

### Customize a Template Using the PPT API

To customize a copy of the default PPT API template or an existing presentation that you want to use as a template:

- 1 Create a `Presentation` object and specify the template for both the presentation argument and the template argument.
- 2 Update the template so that it contains the slides and content that you want in the template. For example, add a slide.
- 3 To save the template, close the `Presentation` object.

For example, this code creates a copy of the default template, adds a title slide to the template, replaces the slide title, and specifies that the title is red:

```
mlreportgen.ppt.Presentation.createTemplate(templateName);
import mlreportgen.ppt.*

templateName = "myTemplate.pptx";
ppt1 = Presentation(templateName,templateName);
open(ppt1);
slide = add(ppt1,"Title Slide");
title = Paragraph("My Title");
title.Style = [title.Style {FontColor("red")}];
replace(slide,"Title",title);
close(ppt1);
rptview(ppt1);
```

When you create a presentation using `myTemplate.pptx` as the template, the presentation has a title slide with the title `My Title` in red.

To customize an existing presentation that you want to use a template, use the same procedure that you use to customize a copy of the default template. For example, this code creates a presentation called `myPresentation.pptx` that has one slide and then customizes `myPresentation.pptx` by replacing the slide title and making the title text red.

```
import mlreportgen.ppt.*

presName = "myPresentation.pptx";
ppt2 = Presentation(presName);
open(ppt2);
slide = add(ppt2,"Title Slide");
close(ppt2);

ppt3 = Presentation(presName,presName);
open(ppt3);
slide = ppt3.Children(1);
title = Paragraph("My Title");
title.Style = [title.Style {FontColor("red")}];
replace(slide,"Title",title);
close(ppt3);
rptview(ppt3);
```

When you create a presentation using `myPresentation.pptx` as the template, the presentation has a title slide with a title `My Title` in red.

## Customize a Template Interactively in PowerPoint

Use PowerPoint for these template customizations:

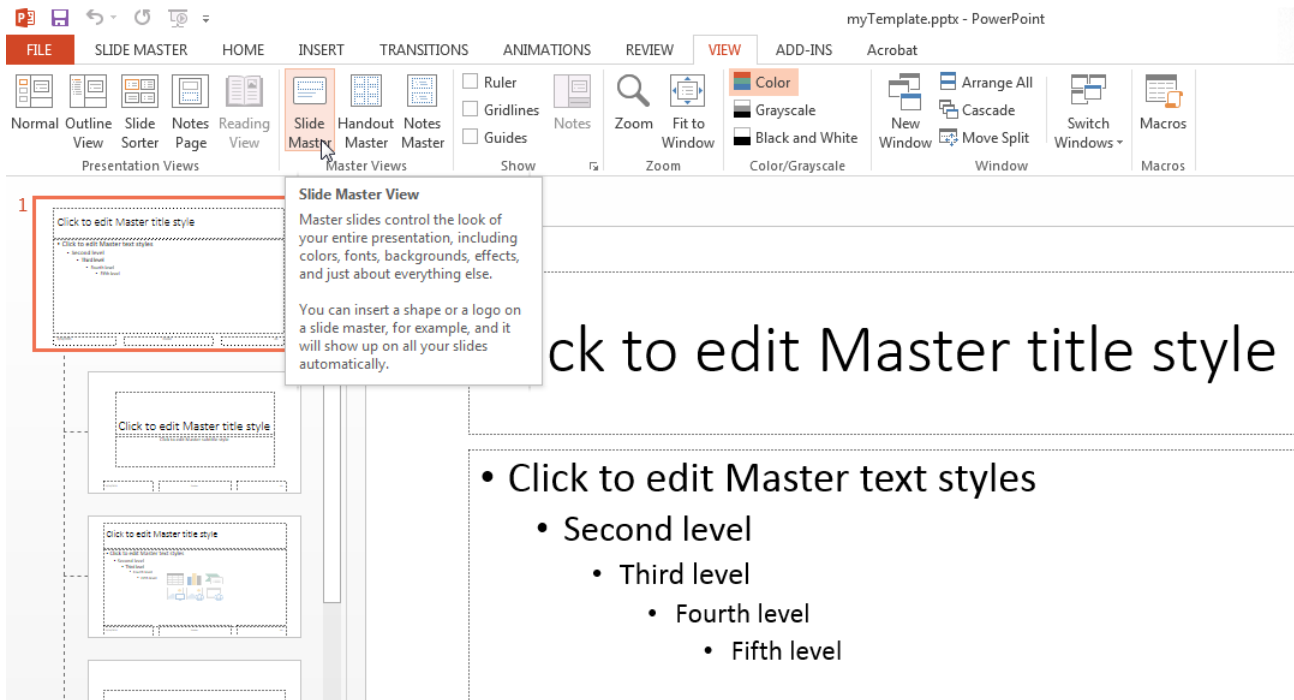
- “Specify the Default Formatting of a Presentation” on page 14-24
- “Add a Slide Master” on page 14-25
- “Format a Slide Layout” on page 14-26
- “Add a Slide Layout” on page 14-28
- “Add a Placeholder to a Slide Layout” on page 14-29

### Specify the Default Formatting of a Presentation

To specify formatting to apply throughout a presentation, use a slide master. The formatting in a slide master is the default formatting for all its child slide layouts.



- 1 In PowerPoint, open a template or a presentation that you want to use as a template.
- 2 In the **View** tab, in the **Master Views** section, click **Slide Master**. For example, using the default PPT API template:



- 3 In a slide master, click in a placeholder. For example, in the master title slide, click in **Click to edit Master title style** text and select a formatting option, such as changing the font color to red.
- 4 Save the template.

### Add a Slide Master

You can add a slide master to a PowerPoint template. Adding a slide master is useful for providing different formatting for different parts of a presentation.

- 1 Open the template in PowerPoint.
- 2 In the **View** tab, in the **Master Views** section, click **Slide Master**.
- 3 In the slide master and layout pane, click after the last slide layout.
- 4 Right-click and select **Insert Slide Master**. A new slide master appears, with a copy of the slide layouts under it.



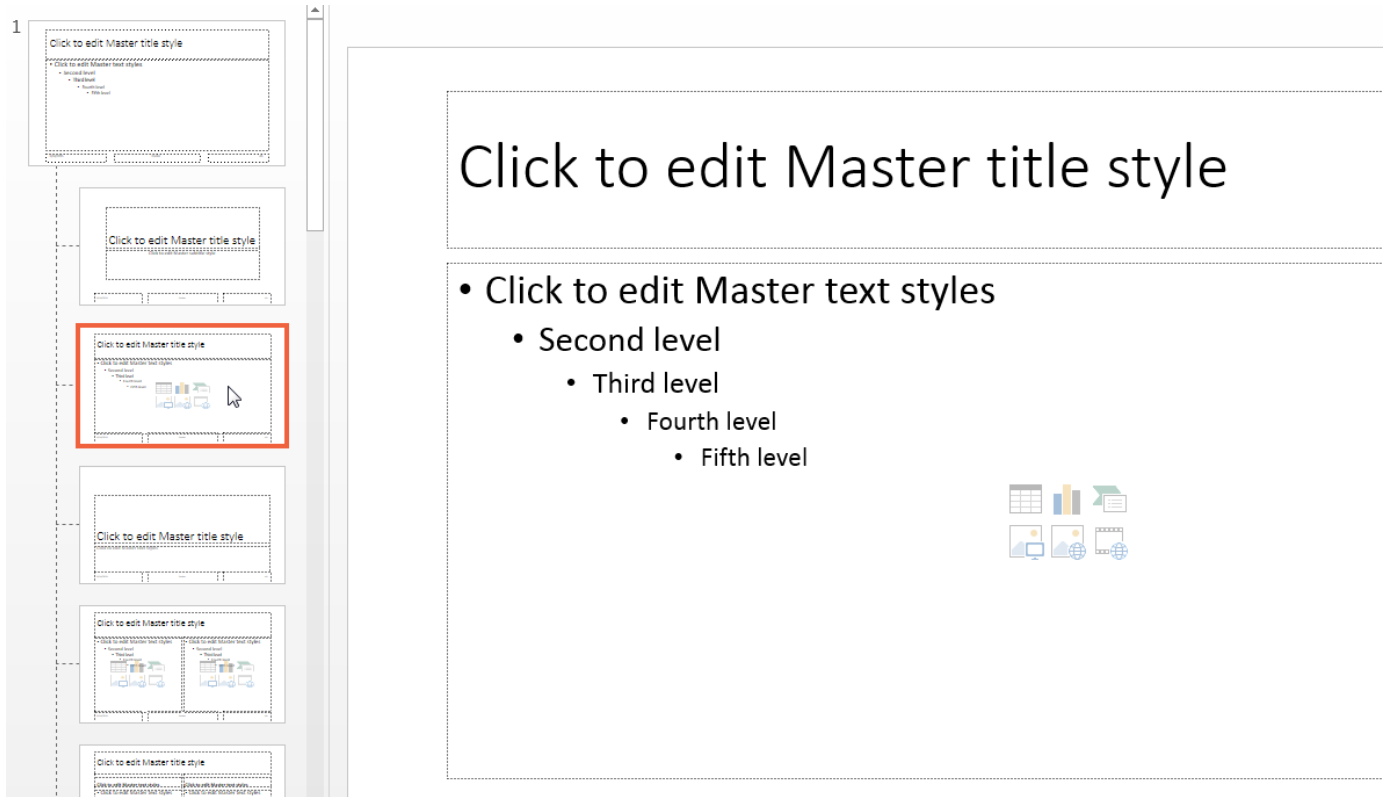
- 5 Format the new slide master.
- 6 Give the slide master a meaningful name. By default, PowerPoint names new masters Custom Design, 1\_Custom Design, 2\_Custom Design, and so on. In the **Slide Master** tab, in the **Edit Master** section, click **Rename** and follow the prompts.
- 7 Save the template.

### Format a Slide Layout

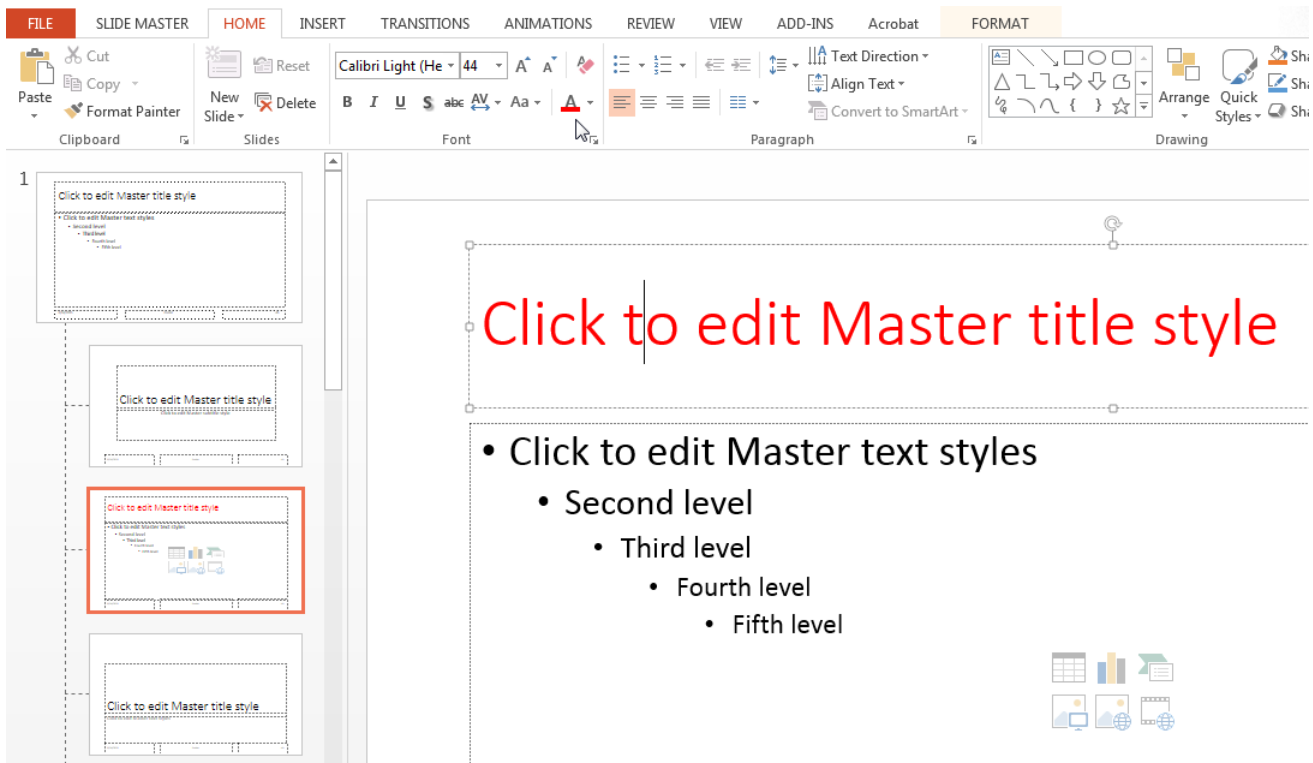
To specify formatting to apply to a specific kind of slide, use a slide layout.

- 1 Open the template in PowerPoint.
- 2 In the **View** tab, in the **Master Views** section, click **Slide Master**.
- 3 From the slide masters and layout pane, select the slide layout whose formatting you want to change. For example, in the PPT API default template, click the Title and Content slide layout.

**Tip** To see the name of a slide layout, point to the layout. A tooltip appears with the name of the slide layout and the number of slides that use the layout.



- 4 In a slide master, click in a placeholder whose formatting you want to change. For example, in the default PPT API template, in the Title and Content slide layout, click in **Click to edit Master title style**. Select a formatting option, such as changing the font color to red. The change applies to the title of that slide layout, but not to the title of other slide layouts.



5 Save the template.

## Add a Slide Layout

You can add a slide layout to a PowerPoint template.

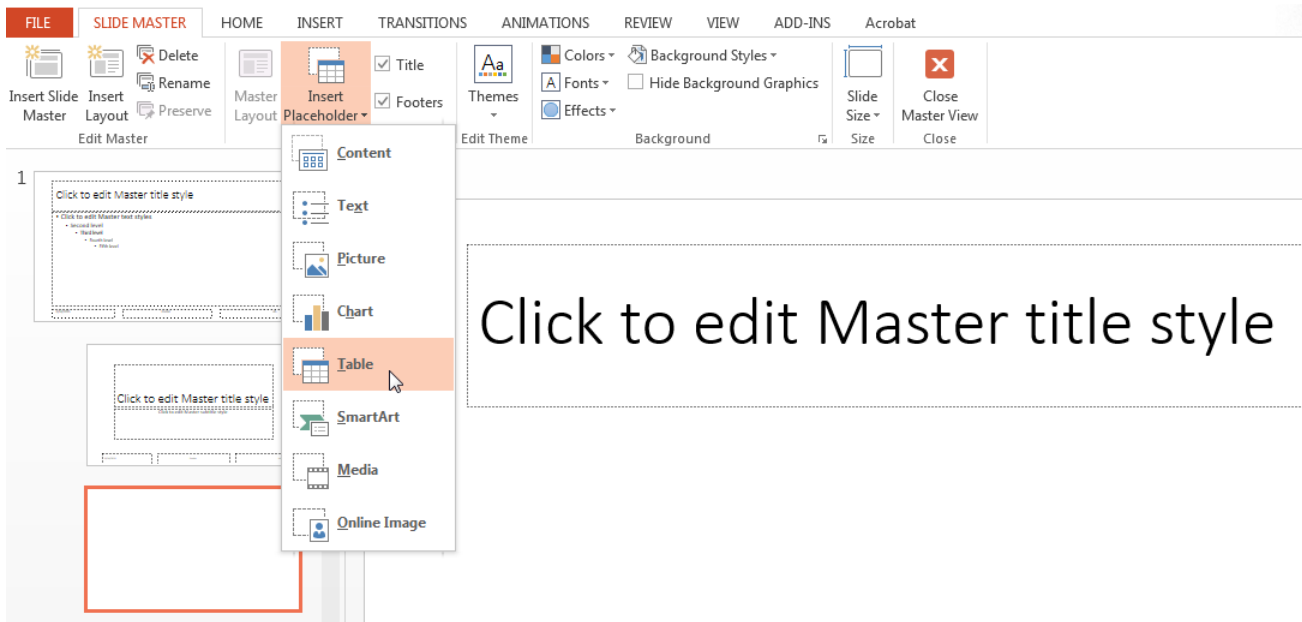
- 1 Open the template in PowerPoint.
- 2 In the **View** tab, in the **Master Views** section, click **Slide Master**.
- 3 In a slide layout, right-click and select **Insert Layout**. A new slide layout appears, with a title placeholder.

---

**Tip** To create a slide layout based on an existing slide layout, right-click in the slide layout that you want to base the layout on. Then select **Duplicate Layout**.

---

- 4 Customize the layout. For example, you can change the font for an existing placeholder or add a placeholder, such as a table placeholder. You can interactively set the location and size of the table placeholder. To remove or add title and footers, use the **Title** and **Footers** check boxes in the **Slide Master** tab.



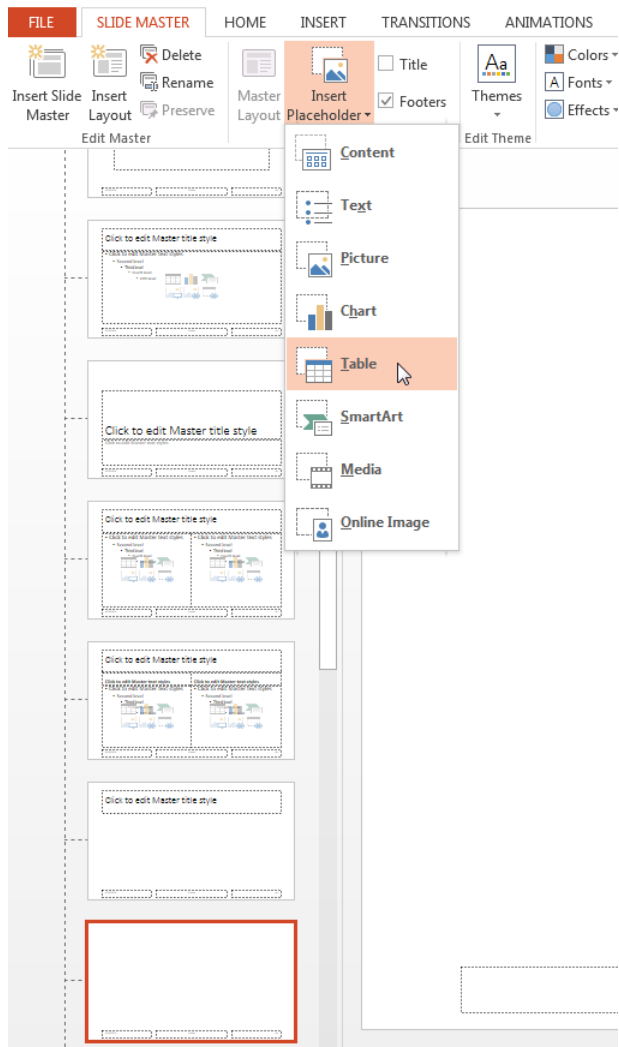
- 5 Give the slide layout a meaningful name. (By default PowerPoint names new layouts Custom Layout, 1\_Custom Layout, 2\_Custom Layout, and so on.) In the **Slide Master** tab, in the **Edit Master** section, click **Rename** and follow the prompts.
- 6 Save the template.

### Add a Placeholder to a Slide Layout

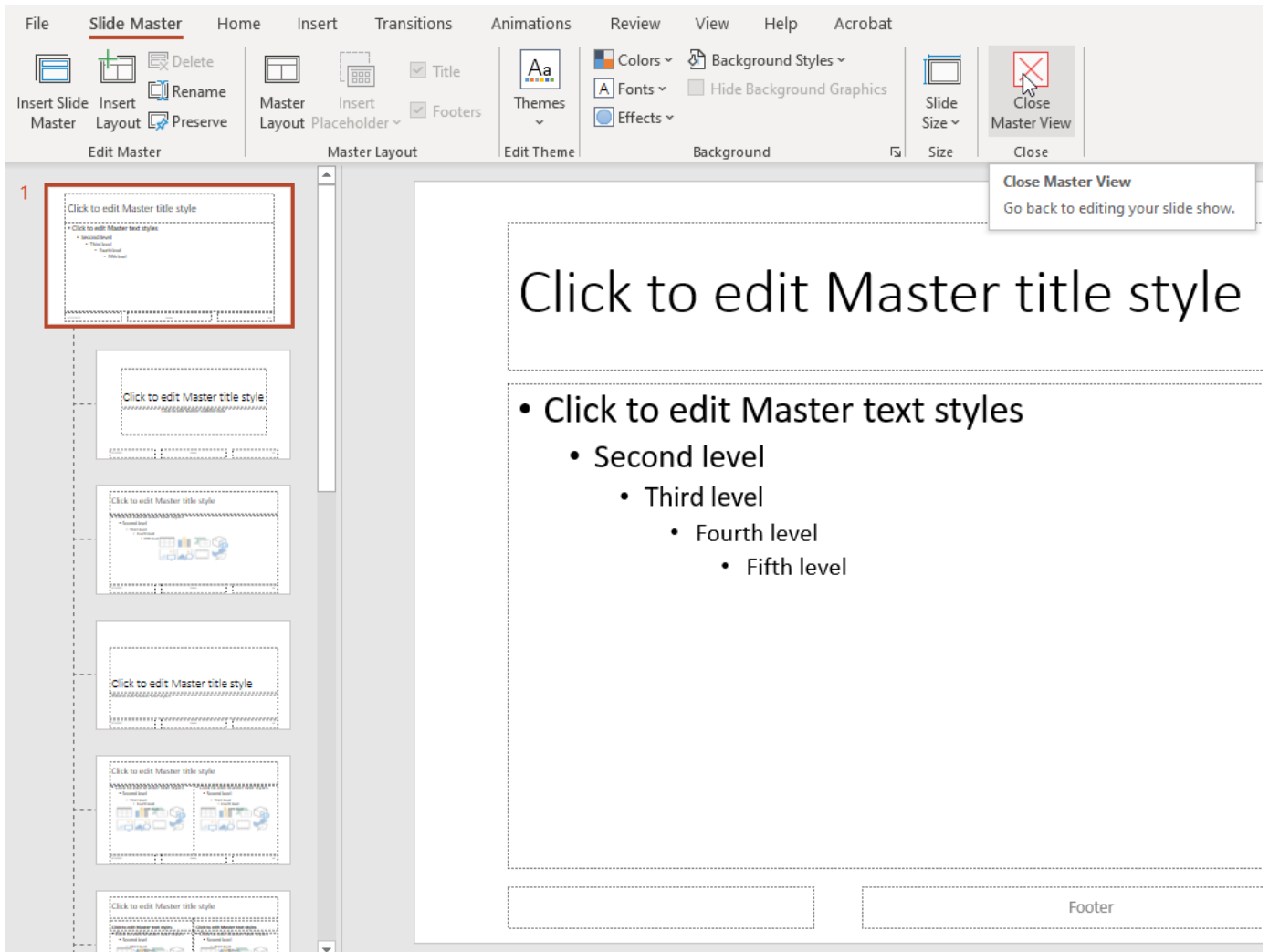
You can add any type of placeholder to any slide layout. However, using the PPT API, you can replace this subset of placeholders:

- Content
- Text
- Picture
- Table

- 1 Open the template in PowerPoint.
- 2 In the **View** tab, in the **Master Views** section, click **Slide Master**.
- 3 In the slide layout pane, select the slide layout to which you want to add the placeholder.
- 4 In the **Slide Master** tab, in the **Master Layout** section, click **Insert Placeholder** and select the type of placeholder from the list. For example, in the default PPT API template, add a Table placeholder to the Blank slide layout.



- 5 In the slide layout, size and position the placeholder.
- 6 Name the placeholders that you want to use when you add or replace content with the PPT API. To name a placeholder, first display the **Selection** pane. On the **Home** tab, in the **Editing** section, select **Select > Selection Pane**. In the **Selection** pane, click the placeholder name and type a new one.
- 7 In the **Slide Master** tab, in the **Close** section, click **Close Master View**.



8 Save the template.

## See Also

### Related Examples

- “Access PowerPoint Template Elements” on page 14-32
- “Presentation Formatting Approaches” on page 14-18

## Access PowerPoint Template Elements

### In this section...

“PPT API Applications and PowerPoint Templates” on page 14-32

“Template Elements” on page 14-32

“View and Change Slide Master Names” on page 14-33

“View and Change Slide Layout Names” on page 14-33

“View and Change Placeholder and Content Object Names” on page 14-34

### PPT API Applications and PowerPoint Templates

The PPT API uses PowerPoint presentations as templates to generate presentations. Templates allow you to specify the fixed content and default layout and appearance of the slides in your presentations. Your MATLAB program can use the PPT API to override the default layout and format of specific slides.

The template can be an empty presentation or a presentation with slides. You can use the following as templates for a PPT API presentation:

- The default PPT API PowerPoint template
- A customized copy of the default PPT API PowerPoint template
- An existing PowerPoint presentation whose content you want to update
- A PowerPoint template that you create or update interactively in PowerPoint

See “Set Up a PowerPoint Presentation Template” on page 14-23.

### Template Elements

PowerPoint templates include several elements that the PPT API uses to generate a presentation. To customize formatting defined in a template, modify one or more of these template elements.

PowerPoint Template Element	Purpose
Slide masters	Applies the slide master formatting globally to the presentation. Specifies a layout and formats common to a set of slide layouts
Slide layouts	Specifies a variant of a slide master layout.
Table styles	Specifies the default appearance of a table. PowerPoint defines a standard set of table styles. You cannot modify these styles but you can use the PPT API to apply these styles to tables you create and override the styles for particular tables.



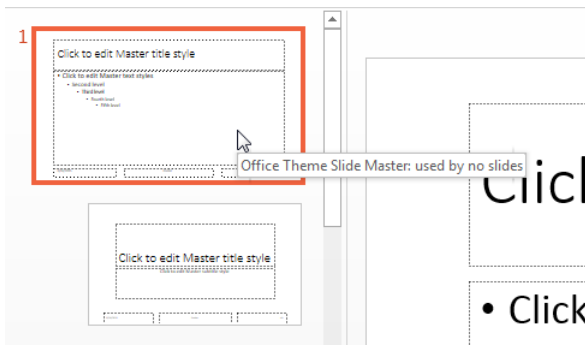
PowerPoint Template Element	Purpose
Placeholders	Specifies an area of a slide layout that you can replace with text, a list, picture, table, or other content. Every placeholder has a name. You can use PowerPoint interactively to assign a name to a placeholder. You can then use the name in your PPT program to replace the placeholder with content.

## View and Change Slide Master Names

A PowerPoint template can have more than one slide master. A slide master can have a child slide layout that has the same name as a child slide layout in another slide master. When you use the PPT API, if the template has multiple slide masters, you need to know the name of the slide master so that you can specify the correct slide layout. You can find out the name in PowerPoint or using the API.

You can rename a master to identify its purpose. You can rename a slide master only in PowerPoint.

- 1 In PowerPoint, select **View > Slide Master**.
- 2 In the slide layout pane, hover over the slide master. Slide masters are numbered and at the top level in the tree hierarchy. A tooltip displays the name. In this figure, **Office Theme** is the name to use in the API. Do not include the text **Slide Master**.



- 3 If you want to rename the master, from the **Slide Master** tab, in the **Edit Master** section, click **Rename** and follow the prompts.

To see slide master names using the PPT API, use the `getMasterNames` method with an `m1reportgen.ppt.Presentation` object. This example uses the default PPT API PowerPoint template, which has one slide master.

```
import m1reportgen.ppt.*;
slides = Presentation('myPresentation');
getMasterNames(slides);
```

```
ans =
```

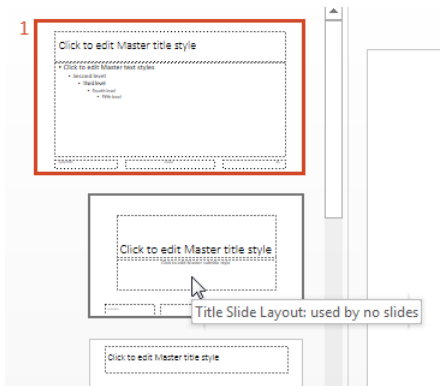
```
    'Office Theme'
```

## View and Change Slide Layout Names

You need to know the name of slide layouts in a PowerPoint template to add a slide using the PPT API. You can find out the slide layout name in PowerPoint and using the API.

When you add a slide layout, you can rename it to identify its purpose. You can rename a slide layout only in PowerPoint.

- 1 In PowerPoint, select **View > Slide Master**.
- 2 In the slide layout pane, hover over a slide layout under a slide master. A tooltip displays the name of the slide layout. In this figure, `Title Slide` is the name to use in the API. Do not include the text `Layout`.



- 3 If you want to rename the slide layout, from the **Slide Master** tab, in the **Edit Master** section, click **Rename** and follow the prompts.

To see slide layout names using the PPT API, use the `Presentation.getLayoutNames` method. You need to get the slide master name before you get the layout names. The PPT API returns slide masters as a cell array. This example uses the default PPT API PowerPoint template to get the slide layouts from the first master in the template.

```
import mlreportgen.ppt.*;
slides = Presentation('myPresentation');
masters = getMasterNames(slides);

layouts = getLayoutNames(slides,masters{1});
layouts
```

Columns 1 through 5

```
'Title Slide' 'Title and Vertica...' 'Vertical Title an...' 'Title and Table' 'Title and Picture'
```

Columns 6 through 11

```
'Title and Content' 'Section Header' 'Two Content' 'Comparison' 'Title Only' 'Blank'
```

Columns 12 through 13

```
'Content with Capt...' 'Picture with Capt...'
```

## View and Change Placeholder and Content Object Names

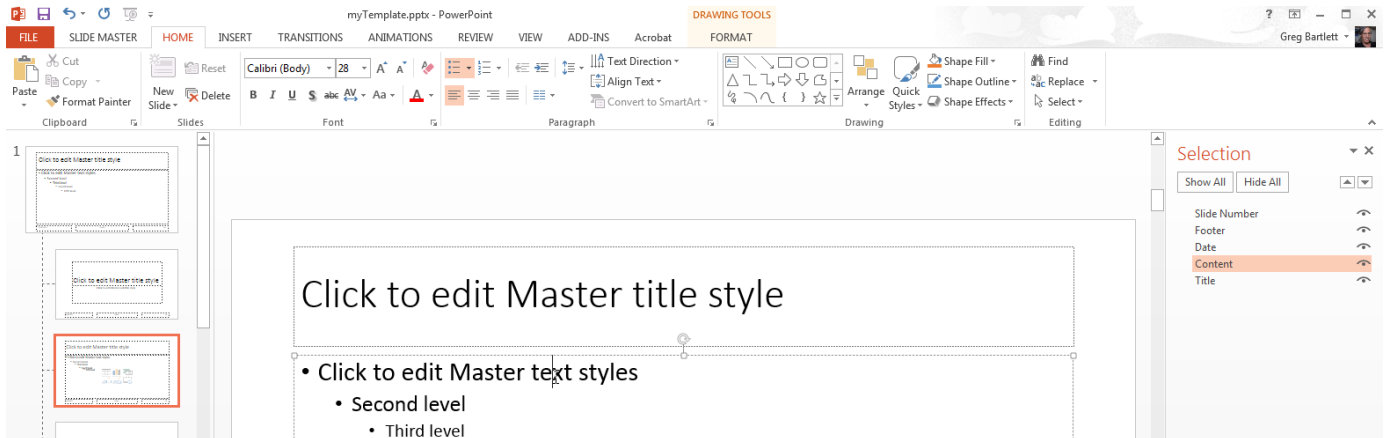
You need to know placeholder names to use the PPT API to replace placeholders with content. You can find out a placeholder name using PowerPoint or using the PPT API.

You can rename a placeholder to identify its purpose.

- 1 In PowerPoint, select **View > Slide Master**.
- 2 In the **Home** tab, in the **Editing** section, select **Select > Selection Pane**.

- 3 In the slide layout pane, select the layout that contains the content placeholder whose name you want to see. The names of the placeholders used in the slide layout appear in the **Selection** pane. Click in a content placeholder to highlight the name in the selection pane.

The figure shows that the name of the content placeholder in the Title and Content slide layout is Content.



- 4 If you want to rename the placeholder, click the name in the **Selection** pane and type a new one.

If you update content in a PowerPoint presentation, to see the name of content objects on that slide, also use the **Selection Pane**. For example:

- 1 Create and generate a presentation with a slide that has a table.

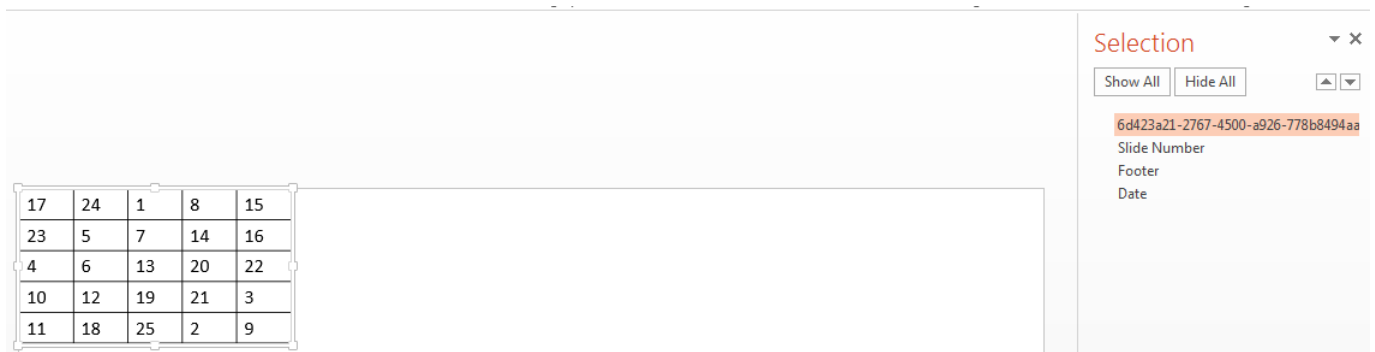
```
import mlreportgen.ppt.*

ppt = Presentation("myTablePresentation.pptx");
open(ppt);

slide1 = add(ppt, 'Blank');
add(slide1, Table(magic(5)));

close(ppt);
rptview(ppt);
```

- 2 In PowerPoint, display the **Selection** pane. The name of the table is a generated string of characters. You can rename it and use the new name with the PPT API.



## **See Also**

### **Related Examples**

- “Set Up a PowerPoint Presentation Template” on page 14-23

### **More About**

- “Presentation Formatting Approaches” on page 14-18

## Define a Style Using Format Objects

A format object is a MATLAB program entity that defines the properties and functions of a specific type of presentation format, such as the weight for text (bold or regular). The PPT API provides a set of constructors for creating several format objects, including:

- `mlreportgen.ppt.Bold` objects
- `mlreportgen.ppt.Italic` objects
- `mlreportgen.ppt.Strike` objects
- `mlreportgen.ppt.Underline` objects
- `mlreportgen.ppt.FontColor` objects

Most PPT API presentation element objects, such as `Text` objects, include a `Style` property that you can set to a cell array of format objects that defines the appearance of the object. For example, to specify the default format for text in a paragraph is red bold text.

```
p = Paragraph("Model Highlights");
p.Style = {FontColor("red"),Bold(true)};
```

You can assign the same array of format objects to more than one PPT API presentation element object. This allows you to create a programmatic equivalent of a template style sheet. For example:

```
import mlreportgen.ppt.*;
ppt = Presentation("myParaPres");
open(ppt);

add(ppt,"Title and Content");
add(ppt,"Title and Content");

caution = {FontColor("red"),Bold(true)};
p1 = Paragraph("Hardware Requirements");
p1.Style = caution;
p2 = Paragraph("Software Requirements");
p2.Style = caution;

titles = find(ppt,"Title");

replace(titles(1),p1);
replace(titles(2),p2);

close(ppt);
```

The PPT API allows you to assign any format object to any presentation object, regardless of whether the format is appropriate for that object type. Format that are not appropriate are ignored.

### See Also

### Related Examples

- “Use Format Properties” on page 14-39
- “Define a Style Using Format Objects” on page 14-37
- “Set Up a PowerPoint Presentation Template” on page 14-23

## **More About**

- “Presentation Formatting Approaches” on page 14-18

## Use Format Properties

### In this section...

"Dot Notation" on page 14-39

"Get the Properties of an Object" on page 14-39

"Set the Properties of an Object" on page 14-40

Most PPT API presentation objects (such as a Paragraph object) include properties that you can use to set the format of the content of an object.

### Dot Notation

To work with PPT API object properties, you use dot notation. Using dot notation involves specifying an object (the variable representing the object) followed by a period and then the property name. For example, suppose that you create a Paragraph object `para1`.

```
para1 = Paragraph("My paragraph");
```

To specify the **Bold** property for the `para1` object, use:

```
para1.Bold = true;
```

### Get the Properties of an Object

To display all the properties of an object that you create, use one of these approaches in MATLAB:

- Omit the semicolon when you create the object.
- Enter the name of the object.

For example, display the properties of the Paragraph object `para1`.

```
para1 = Paragraph("My paragraph")
```

```
para1 =
```

```
Paragraph with properties:
```

```

    Bold: []
  FontColor: []
    Italic: []
    Strike: []
  Subscript: []
  Superscript: []
    Underline: []
    Level: []
    Style: []
  Children: [1x1 mlreportgen.ppt.Text]
    Parent: []
    Tag: 'ppt.Paragraph:22'
    Id: '22'
```

To display the value of a specific property, such as the **Bold** property, use dot notation, without a semicolon.

```
par1 = Paragraph("My paragraph");  
par1.Bold  
  
ans =  
  
    []
```

## Set the Properties of an Object

```
par1 = Paragraph("My paragraph"); par1.Bold = true;
```

You can set some PPT API object properties using the object constructor. The PPT API sets other properties. For most PPT API objects, you can change the values of properties that you specified in the constructor. Also, you can specify values for additional properties.

To specify a value for an object property, use dot notation. For example, to set the default for text in the `par1` paragraph to bold:

```
par1 = Paragraph("My paragraph");  
par1.Bold = true;
```

For some presentation objects, you can use the `Style` property to specify formatting options that are not available in the other properties of the object. For example, a `TableEntry` object does not have a `Bold` property. However, you can specify bold as the default for text in the `TableEntry` by using the `Style` property of the `TableEntry` object.

```
te = TableEntry();  
te.Style = {Bold(true)};
```

## See Also

### Related Examples

- “Define a Style Using Format Objects” on page 14-37
- “Set Up a PowerPoint Presentation Template” on page 14-23

### More About

- “Presentation Formatting Approaches” on page 14-18



## Update Presentation Content Programmatically

### In this section...

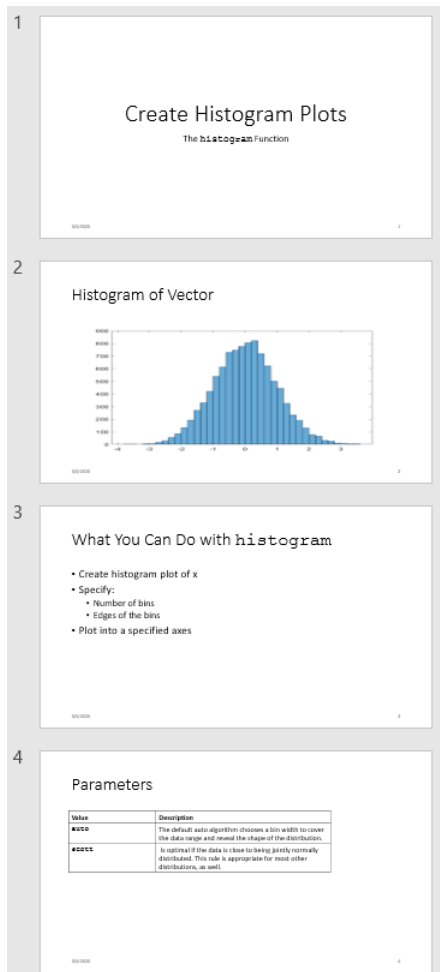
“Generate the Existing Presentation” on page 14-41  
“Updates to the Presentation” on page 14-42  
“Set Up the Existing Presentation” on page 14-43  
“Import the PPT API Package” on page 14-44  
“Create the Presentation Object” on page 14-44  
“Replace a Picture” on page 14-45  
“Replace Text with Links” on page 14-45  
“Replace a Table” on page 14-45  
“Insert a New Slide” on page 14-46  
“Generate and View the Presentation” on page 14-46  
“Code for myUpdatedPresentation” on page 14-46

You can use the PPT API to update content programmatically in an existing PowerPoint presentation.

### Generate the Existing Presentation

This example updates content in a PowerPoint presentation `myPresentation`. To generate the presentation, run the example in “Create a Presentation Programmatically” on page 14-49. Although you create the presentation programmatically, after you generate it, the presentation is like any other PowerPoint presentation. The presentation includes four slides:

- Title slide with the title `Create Histogram Plots`
- A histogram of a vector
- Slide with the title `What You Can Do with histogram`
- Histogram function parameters



To use the PPT API to update content in an existing PowerPoint presentation programmatically, you:

- Set up the PowerPoint presentation by naming content objects that you want to replace. If you want to add new content, insert placeholders in the presentation for that content.
- In MATLAB, import the `mlreportgen.ppt` PPT API package.
- Create a `Presentation` object that uses the existing presentation as the template for updated version.
- Replace any existing slide content that you want to update.
- Add slides any new slides.
- Generate the presentation.

## Updates to the Presentation

In this example, you use the PPT API to make these changes to the `myPresentation` presentation:

- Replace the picture on the second slide.
- Replace the text on the third slide.
- Replace the table on the fourth slide.

- Insert a new slide before the slide with the plot.

Here is the updated presentation:

1

Create Histogram Plots  
The `histogram` Function

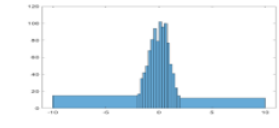
2

Histogram Plots

- You can use the `histogram` function to create many types of plots.

3

Histogram with Specified Bin Edges



4

Related Functions

- [histcounts](#)
- [linspace](#)

5

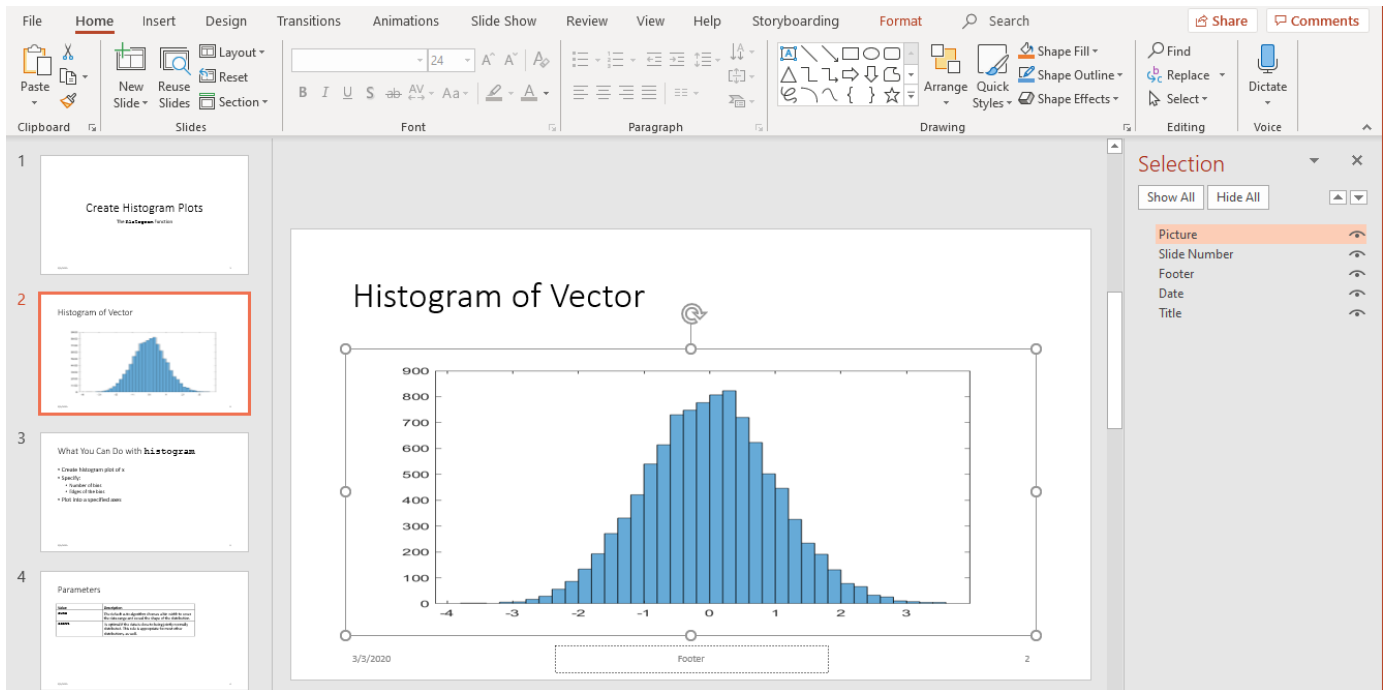
Parameters

Long Name	Short Name	Default Value
<code>edges</code>	<code>E</code>	[1 1 0]
<code>weights</code>	<code>W</code>	[1 1 1]

## Set Up the Existing Presentation

A PPT API program uses a PowerPoint template to generate a presentation. When you update an existing PowerPoint presentation programmatically, use that presentation as the template for the updated presentation. To update content in the Slide objects, use the PPT API.

- 1 Open the myPresentation presentation. In PowerPoint, click **View > Normal**.
- 2 View the names of content objects in the slides. In the **Home** tab, click **Select > Selection Pane**. When you click content in a slide, the **Selection** pane highlights the name of the content object.



- 3 Rename content objects. In the PowerPoint **Selection** pane, click in the content name box and replace the current name with the name you want. Use these unique names to update content objects.
  - In the second slide, change the Title object name to Histogram and the Picture object name to HistBins.
  - In the third slide, change Title to RelatedFuncs. Change Content to FuncList.
  - In the fourth slide, change Table to ParamTable.

## Import the PPT API Package

All PPT API class names include the prefix `mlreportgen.ppt`. To avoid the need to include the prefix in your code, insert this statement at the beginning of a PPT API program.

```
import mlreportgen.ppt.*
```

---

**Note** The `import` line is the first line in the example program. This example creates a PPT API program in sections and therefore does not show the `import` command. To view the complete program, click `myUpdatedPresentation` program on page 14-46.

---

## Create the Presentation Object

Create a Presentation object. Specify:

- `myUpdatedPresentation.pptx` as the output file for the generated presentation.
- `myPresentation.pptx` as the PowerPoint template. Use the presentation file that you want to update as the template file.

```
ppt = Presentation('myUpdatedPresentation.pptx', 'myPresentation.pptx');
open(ppt);
```

Specifying a different name for the output file preserves the original presentation. If you want to overwrite the existing presentation, you can use the template file name as the file name for the output file.

## Replace a Picture

Change the title of the second slide. Create a `Picture` object to replace the existing picture. You can use a `find` method with the `Presentation` object to find content objects named `HistBins` and `Histogram` (the unique names you specified using PowerPoint).

```
histTitle = Paragraph('Histogram with Specified Bin Edges');
replace(ppt, 'Histogram', histTitle);

x = randn(1000,1);
edges = [-10 -2:0.25:2 10];
h = histogram(x,edges);
saveas(gcf, 'hist_plot.png');

plotEdges = Picture('hist_plot.png');

replace(ppt, 'HistBins', plotEdges);
```

## Replace Text with Links

Change the title of the third slide. Create text to replace the existing text. The text includes links to the MathWorks online documentation. Append `ExternalLink` objects to `Paragraph` objects, and replace the slide content using a cell array of the `Paragraph` objects.

```
funcsTitle = Paragraph('Related Functions');
replace(ppt, 'RelatedFuncs', funcsTitle);

histCounts = Paragraph();
histCountsLink = ExternalLink...
('https://www.mathworks.com/help/matlab/ref/histcounts.html', 'histcounts');
append(histCounts, histCountsLink);

fewerbins = Paragraph();
fewerbinsLink = ExternalLink...
('https://www.mathworks.com/help/matlab/ref/matlab.graphics.chart.primitive.histogram.fewerbins.html', 'fewerbins');
append(fewerbins, fewerbinsLink);

replace(ppt, 'FuncList', {histCounts, fewerbins});
```

## Replace a Table

To create a table, create a `Table` object. In the `Table` constructor, you can specify a cell array of values for the table cells. To get bold text for the top row, include `Paragraph` objects as the first three elements of the cell array. Then replace the table.

```
long = Paragraph('Long Name');
long.Bold = true;
short = Paragraph('Short Name');
short.Bold = true;
rgb = Paragraph('RGB triplet');
rgb.Bold = true;

table2 = Table({long, short, rgb; 'yellow', 'y', '[1 1 0]'; 'green', 'g', '[1 0 1] '});

contents = find(ppt, 'ParamTable');
replace(ppt, 'ParamTable', table2);
```

## Insert a New Slide

You can use the PPT API to insert a new slide in an existing presentation and you can specify the numerical location of the slide. For example, this code makes a new slide the fifth slide in a presentation.

```
newSlide = add(ppt, 'Title and Content',5);
```

However, to have a slide precede a specific slide, even if later you add or remove other slides, you can specify a reference slide. To use this approach when updating an existing PowerPoint presentation, use the PPT API to name the reference slide. Use the name of the reference slide when you insert a new slide.

```
ppt.Children(2).Name = 'ReferenceSlide';

refSlide = find(ppt, 'ReferenceSlide');
introSlide = add(ppt, 'Title and Content', refSlide);

contents = find(introSlide, 'Title');
replace(contents(1), 'Histogram Plots');

introText = Paragraph('You can use the ');
code = Text('histogram');
code.Font = 'Courier New';
append(introText, code);
append(introText, ' function to create many types of plots.');
```

```
contents = find(introSlide, 'Content');
replace(contents(1), introText);
```

## Generate and View the Presentation

Generate the PowerPoint presentation. Use a close method with a Presentation object. View the presentation.

```
close(ppt);
rptview(ppt);
```

## Code for myUpdatedPresentation

Here is the complete PPT API program to create the myUpdatedPresentation presentation.

---

**Note** This code requires that the myPresentation.pptx file be in your current folder. To generate that presentation, run the example in “Create a Presentation Programmatically” on page 14-49. Before you run the code for myUpdatedPresentation, be sure that the existing presentation includes the changes described in “Set Up the Existing Presentation” on page 14-43.

---

```
import mlreportgen.ppt.*;

ppt = Presentation('myUpdatedPresentation.pptx', 'myPresentation.pptx');
open(ppt);

histTitle = Paragraph('Histogram with Specified Bin Edges');
replace(ppt, 'Histogram', histTitle);

x = randn(1000,1);
```

```

edges = [-10 -2:0.25:2 10];
h = histogram(x,edges);
saveas(gcf,'hist_plot.png');

plotEdges = Picture('hist_plot.png');

replace(ppt,'HistBins',plotEdges)

funcsTitle = Paragraph('Related Functions');
replace(ppt,'RelatedFuncs',funcsTitle);

histCounts = Paragraph();
histCountsLink = ExternalLink...
('https://www.mathworks.com/help/matlab/ref/histcounts.html','histcounts');
append(histCounts,histCountsLink);

fewerbins = Paragraph();
fewerbinsLink = ExternalLink...
('https://www.mathworks.com/help/matlab/ref/matlab.graphics.chart.primitive.histogram.fewerbins.html','fewerbins');
append(fewerbins,fewerbinsLink);

replace(ppt,'FuncList',{histCounts,fewerbins});

long = Paragraph('Long Name');
long.Bold = true;
short = Paragraph('Short Name');
short.Bold = true;
rgb = Paragraph('RGB triplet');
rgb.Bold = true;

table2 = Table({long,short,rgb;'yellow','y','[1 1 0]'; 'green', 'g','[1 0 1] '});

contents = find(ppt,'ParamTable');
replace(ppt,'ParamTable',table2);

ppt.Children(2).Name = 'ReferenceSlide';

refSlide = find(ppt,'ReferenceSlide');
introSlide = add(ppt,'Title and Content',refSlide(1));

contents = find(introSlide,'Title');
replace(contents(1),'Histogram Plots');

introText = Paragraph('You can use the ');
code = Text('histogram ');
code.Style = {FontFamily('Courier New')};
append(introText,code);
append(introText,'function to create many types of plots.');
```

```

contents = find(introSlide,'Content');
replace(contents(1),introText);

close(ppt);
rptview(ppt);

```

## See Also

### Related Examples

- “Create a Presentation Programmatically” on page 14-49
- “Set Up a PowerPoint Presentation Template” on page 14-23
- “Access PowerPoint Template Elements” on page 14-32
- “Add Slides” on page 14-56
- “Create and Format Text” on page 14-65
- “Create and Format Paragraphs” on page 14-67
- “Create and Format Tables” on page 14-69
- “Create and Format Pictures” on page 14-76

- “Create and Format Links” on page 14-78



## Create a Presentation Programmatically

This example shows how to create a PowerPoint® presentation by using the MATLAB® API for PowerPoint (PPT API). The example generates these slides:

1

# Create Histogram Plots

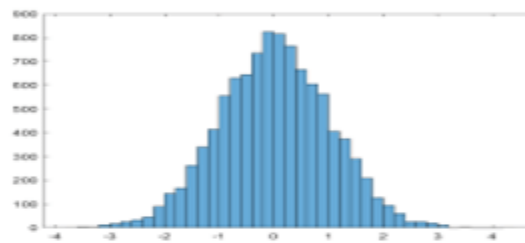
The `histogram` Function

3/13/2018

1

2

## Histogram of Vector



3/13/2018

2

3

## What You Can Do with `histogram`

- Create histogram plot of `x`
- Specify:
  - Number of bins
  - Edges of the bins
- Plot into a specified axes

3/13/2018

3

4

## Parameters

Value	Description
<code>auto</code>	The default auto algorithm chooses a bin width to cover the data range and reveal the shape of the distribution.
<code>scott</code>	is optimal if the data is close to being jointly normally distributed. This rule is appropriate for most other distributions, as well.

3/13/2018

4

To programmatically create a presentation:

- 1 Import the PPT API package.
- 2 Create a presentation container.
- 3 Add slides.
- 4 Add content to the slides.
- 5 Generate the presentation.

### Import the PPT API Package

The PPT API classes belong to the `mlreportgen.ppt` package. Import this package so that you do not have to include the package name when you call the PPT API object constructors and methods.

```
import mlreportgen.ppt.*
```

### Create a Presentation Container

Create an `mlreportgen.ppt.Presentation` object to contain the presentation. For this example, specify that the output file name is `myPresentation` and do not specify a template.

```
ppt = Presentation('myPresentation.pptx');
```

Because you did not specify a template, the PPT API uses the default template. A template defines the default slide layouts and styles. To create a custom template, see “Set Up a PowerPoint Presentation Template” on page 14-23. You can override the default style defined by a template by using format properties and objects. See “Presentation Formatting Approaches” on page 14-18.

### Add Slides and Slide Content

To add a slide, use the `add` method and specify a slide layout that is available in the template. See “Set Up a PowerPoint Presentation Template” on page 14-23. This example uses these slide layouts that are included with the default template:

- Title Slide
- Title and Picture
- Title and Content
- Title and Table

To add content to a slide, use the `replace` method to replace content placeholders with the new content. For more information about adding and replacing content, see “Add and Replace Presentation Content” on page 14-58. To use the `replace` method, you must specify the name that identifies a placeholder in the slide layout. For example, the `Title Slide` layout has a `Title` placeholder and a `Subtitle` placeholder. For information about how to find the content placeholder names for a particular slide layout, see “Access PowerPoint Template Elements” on page 14-32.

### Add a Title Slide

To add a title slide, use the `Title Slide` layout.

```
titleSlide = add(ppt, 'Title Slide');
```

The `Title Slide` layout has these placeholders:

- Title
- Subtitle

Replace the Title placeholder with the title text.

```
replace(titleSlide, 'Title', 'Create Histogram Plots');
```

Build the title text in pieces so that you can format the histogram function name in a different font.

```
subtitleText = Paragraph('The ');  
funcName = Text('histogram');  
funcName.Font = 'Courier New';
```

```
append(subtitleText, funcName);  
append(subtitleText, ' Function');
```

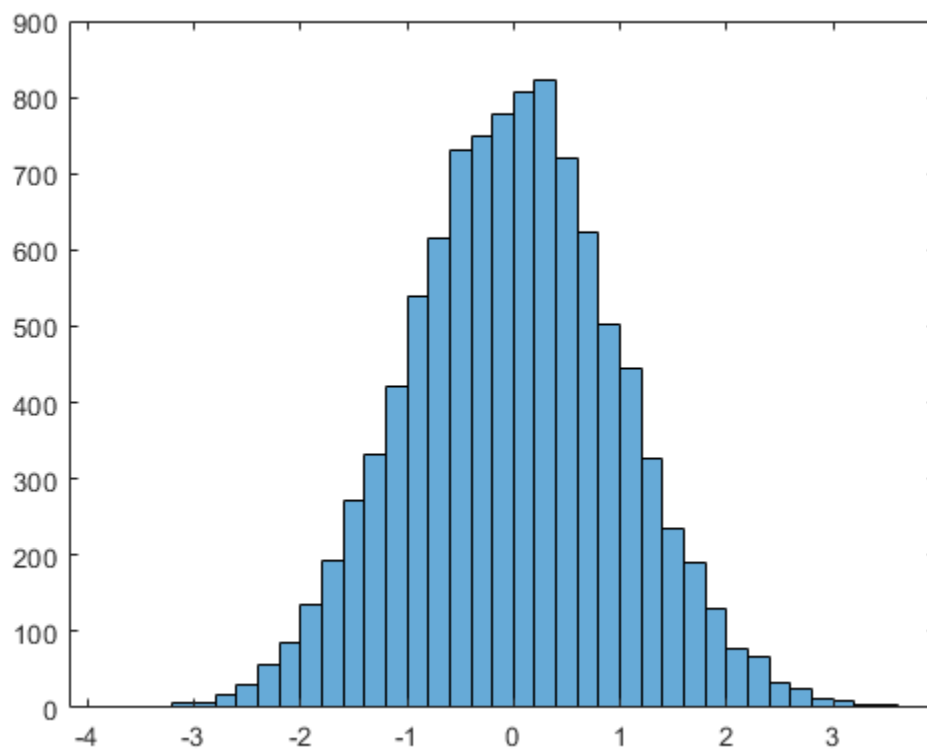
Replace the Subtitle placeholder with the text contained in SubtitleText.

```
replace(titleSlide, 'Subtitle', subtitleText);
```

### Add a Slide with a Picture

Create an image file to use for the slide picture.

```
x = randn(10000,1);  
h = histogram(x);  
saveas(gcf, 'myPlot_img.png');
```



Create an `mlreportgen.ppt.Picture` object from the image file.

```
plot1 = Picture('myPlot_img.png');
```

Add a picture slide to the presentation by using a `Title` and `Picture` layout.

```
pictureSlide = add(ppt, 'Title and Picture');
```

The `Title` and `Picture` layout has these placeholders:

- `Title`
- `Picture`

Replace the `Title` placeholder with the title text and the `Picture` placeholder with `plot1`.

```
replace(pictureSlide, 'Title', 'Histogram of Vector');
replace(pictureSlide, 'Picture', plot1);
```

The contents of an image file are copied into the presentation when the presentation is closed. Do not delete or overwrite the image file before the presentation is closed. If your presentation program creates multiple image files, give them unique file names.

### Add a Slide with Text

To add a slide with text, use the `Title` and `Content` layout.

```
textSlide = add(ppt, 'Title and Content');
```

The `Title` and `Content` layout has these placeholders:

- `Title`
- `Content`

Build the title text in pieces so that you can format the `histogram` function name in a different font.

```
titleText = Paragraph('What You Can Do with ');
func = Text('histogram');
func.Font = 'Courier New';
append(titleText, func);
```

Replace the `Title` and `Content` placeholders.

```
replace(textSlide, 'Title', titleText);
replace(textSlide, 'Content', {'Create histogram plot of x', ...
'Specify:', {'Number of bins', 'Edges of the bins'}, ...
'Plot into a specified axes'});
```

### Add a Slide with a Table

To add a slide with a table, use the `Title` and `Table` layout.

```
tableSlide = add(ppt, 'Title and Table');
```

The `Title` and `Table` layout has these placeholders:

- `Title`
- `Table`

Replace the Title placeholder.

```
replace(tableSlide, 'Title', 'Parameters');
```

You can use several approaches to create a table. See “Create and Format Tables” on page 14-69. This example builds a table row by row.

- 1 Create a table as an `mlreportgen.ppt.Table` object.
- 2 Create an `mlreportgen.ppt.TableRow` object for each row of the table.
- 3 Create `mlreportgen.ppt.TableEntry` objects and append them to the table rows.

```
paramTable = Table();
colSpecs(2) = ColSpec('6in');
colSpecs(1) = ColSpec('3in');
paramTable.ColSpecs = colSpecs;

tr1 = TableRow();
tr1.Style = {Bold(true)};

tr1te1Text = Paragraph('Value');
tr1te2Text = Paragraph('Description');
tr1te1 = TableEntry();
tr1te2 = TableEntry();
append(tr1te1, tr1te1Text);
append(tr1te2, tr1te2Text);
append(tr1, tr1te1);
append(tr1, tr1te2);

tr2 = TableRow();
tr2te1Text = Paragraph('auto');
tr2te1Text.Font = 'Courier New';
tr2te2Text = Paragraph('The default auto algorithm chooses a bin width to ');
append(tr2te2Text, 'cover the data range and reveal the shape of the distribution. ');
tr2te1 = TableEntry();
tr2te2 = TableEntry();
append(tr2te1, tr2te1Text);
append(tr2te2, tr2te2Text);
append(tr2, tr2te1);
append(tr2, tr2te2);

tr3 = TableRow();
tr3te1Text = Paragraph('scott');
tr3te1Text.Font = 'Courier New';
tr3te2Text = Paragraph(' Is optimal if the data is close ');
append(tr3te2Text, 'to being jointly normally distributed. This rule is ');
append(tr3te2Text, 'appropriate for most other distributions, as well. ');
tr3te1 = TableEntry();
tr3te2 = TableEntry();
append(tr3te1, tr3te1Text);
append(tr3te2, tr3te2Text);
append(tr3, tr3te1);
append(tr3, tr3te2);

append(paramTable, tr1);
append(paramTable, tr2);
append(paramTable, tr3);
```

Replace the Table placeholder with `paramTable`.

```
replace(tableSlide, 'Table', paramTable);
```

### **Generate and View the Presentation**

```
close(ppt);  
rptview(ppt);
```

### **See Also**

`mlreportgen.ppt.Presentation` | `mlreportgen.ppt.Slide` | `mlreportgen.ppt.Paragraph`  
| `mlreportgen.ppt.Text` | `mlreportgen.ppt.Table` | `mlreportgen.ppt.TableRow` |  
`mlreportgen.ppt.TableEntry`

### **More About**

- “Generate a Presentation From the Results of a MATLAB Application” on page 14-84
- “Set Up a PowerPoint Presentation Template” on page 14-23
- “Access PowerPoint Template Elements” on page 14-32
- “Add Slides” on page 14-56
- “Update Presentation Content Programmatically” on page 14-41
- “Create and Format Text” on page 14-65
- “Create and Format Paragraphs” on page 14-67
- “Create and Format Tables” on page 14-69
- “Create and Format Pictures” on page 14-76

## Add Slides

### In this section...

“Specify the Order of a Slide” on page 14-56

“Specify the Slide Master” on page 14-57

To add a slide to a presentation, use the PPT API to add slide based on a slide layout defined in the PowerPoint presentation template. If the template does not include slide layout that meets your requirements, you can add a slide layout. For details, see “Add a Slide Layout” on page 14-28.

To add a slide, use the `add` method with an `mlreportgen.ppt.Presentation` object. For example, using the default PPT API template, you can add a slide using the Title and Content slide layout.

```
import mlreportgen.ppt.*;
ppt = Presentation('myPresentation');
slide1 = add(ppt, 'Title and Content');
```

When you add a slide, the PPT API creates an `mlreportgen.ppt.Slide` object. However, you cannot add a slide by using a `Slide` constructor.

### Specify the Order of a Slide

By default, the order in which you add slides in a PPT API program determines the order in which the slides appear. For example, this code makes the `titleSlide` slide the first slide in the presentation. The `contentSlide` slide is the second slide.

```
ppt = Presentation('myPresentation');
titleSlide = add(ppt, 'Title Slide');
contentSlide = add(ppt, 'Title and Content');
```

When you add a slide, to specify explicitly the order in which it appears, you can:

- Specify the slide the new slide precedes. This approach is useful to keep slides together as you add or delete slides.
- Specify an index indicating the numerical position of the slide in the presentation. This approach is useful when you want a slide to appear always in the same numerical position.

The first approach places the new slide immediately before slide you specify. If you created the reference slide using the PPT API, you can specify the `Slide` object. For example, using the default PPT API template, this code causes the `pictureSlide` to appear immediately before the `introSlide`.

```
ppt = Presentation('myPresentation');
titleSlide = add(ppt, 'Title Slide');
introSlide = add(ppt, 'Title Slide');
pictureSlide = add(ppt, 'Title and Picture', introSlide);
```

In a presentation created using PowerPoint, adding a slide immediately before a slide that you created using PowerPoint requires a few steps.

- 1 In PowerPoint, identify the position of the reference slide you want the new slide to precede.



- 2 Open the PPT API program and give a name to the reference slide you want to position the new slide before. For example, assume that the reference slide is the second slide in a PowerPoint presentation.

```
ppt = Presentation('myPresentation', 'myPresentation');
open(ppt);

ppt.Children(2).Name = 'ReferenceSlide';
close(ppt);
```

- 3 To identify the reference slide object, use the slide name. Add the new slide relative to the reference slide.

```
ppt = Presentation('myPresentation', 'myPresentation');
open(ppt);

refSlide = find(ppt, 'ReferenceSlide');
add(ppt, 'Blank', refSlide);

close(ppt);
```

To use the second approach, specify an index representing the numerical position for the slide. For example, using the default PPT API template, this code makes `pictureSlide` the second slide in the presentation.

```
ppt = Presentation('myPresentation');

titleSlide = add(ppt, 'Title Slide');
introSlide = add(ppt, 'Title and Content');
pictureSlide = add(ppt, 'Title and Picture', 2);
```

## Specify the Slide Master

A template can have multiple slide masters. Two or more slide masters can have a child slide layout with the same name. By default, when you specify the slide layout using PPT API, the API uses the first slide layout that has the name you specify. If you specify a slide master in an `add` method, specify the slide master argument immediately after the slide layout argument. For example, this code uses the `Title Slide` slide layout that is a child of the `myCustomMaster` slide master.

```
ppt = Presentation('myPresentation');
titleSlide = add(ppt, 'Title Slide', myCustomMaster);
```

## See Also

### Functions

`add` | `getMasterNames` | `getLayoutNames`

## Related Examples

- “Create a Presentation Object to Hold Content” on page 14-12
- “Add and Replace Presentation Content” on page 14-58

## Add and Replace Presentation Content

In this section...
“Set Up the Template” on page 14-58
“Replace Content” on page 14-58
“Add and Replace Text” on page 14-59
“Add or Replace a Table” on page 14-61
“Add or Replace a Picture” on page 14-62

To use the PPT API to add, or replace, content in a PowerPoint presentation:

- Set up a PowerPoint template to hold the presentation content you want to add or replace.
- Create PPT API content objects, such as **Paragraph**, **Table**, and **Picture** objects.
- Use PPT API content objects to add or replace presentation content.

You can add and replace content in several ways. For example, you can:

- Add or replace content globally in a presentation or locally in a specific slide.
- Add content to a text box.
- Replace a text box, table, or picture with content of the same type.
- Replace a placeholder with content corresponding to that placeholder.

You cannot replace part of a paragraph, table, or text box. Replace the whole content object.

### Set Up the Template

You can replace or add content to an existing PowerPoint presentation without modifying the template. However, using the PPT API requires knowledge of template and slide objects, including:

- Slide master names
- Slide layout names
- Slide placeholder and content object names
- Table style names

You can use using PowerPoint to add placeholders to a presentation and then use the PPT API to replace the placeholder with content. To replace a specific content object in a presentation, you can use PowerPoint to give a unique name to the content object. Then use that name with the PPT API.

For more information about using PowerPoint templates with a PPT API program, see:

- “Set Up a PowerPoint Presentation Template” on page 14-23
- “Access PowerPoint Template Elements” on page 14-32

### Replace Content

You can replace content by specifying the content object name in a `replace` method with a `Slide` object. For example, in the default PPT API template, the Title Slide layout has a content object called `Title`.

```
titleSlide = add(ppt, 'Title Slide');
replace(titleSlide, 'Title', 'This Is My Title');
```

To replace presentation content, you can use a `find` method with a `Presentation` or `Slide` object. The `find` method searches for content objects whose `Name` property value matches the search value you specify. Then you can use the index of the returned item that you want to update.

```
ppt = Presentation('myPresentation');
titleSlide = add(ppt, 'Title Slide');

contents = find(ppt, 'Title');
replace(contents(1), 'This Is My Title');
```

## Add and Replace Text

You can use these approaches to add or replace text in a presentation.

Text Specification Technique	Associated PPT API Objects
Specify text as part of creating these objects.	<ul style="list-style-type: none"> <li>Text</li> <li>Paragraph</li> <li>ExternalLink</li> <li>InternalLink</li> </ul>
Append text to a paragraph.	Append text to these PPT API objects: <ul style="list-style-type: none"> <li>Paragraph</li> <li>TableEntry</li> </ul>
Replace a Paragraph object in a presentation or slide.	Specify a character vector, Paragraph object, or a cell array of character vectors or Paragraph objects or a combination of both kinds of objects, for the <code>replace</code> method with these objects: <ul style="list-style-type: none"> <li>Presentation</li> <li>Slide</li> </ul>
Add to or replace text in a placeholder object.	<ul style="list-style-type: none"> <li>Add to a <code>ContentPlaceholder</code> object a character vector, Paragraph object, or with a cell array of character vectors or Paragraph objects, or a combination of both.</li> <li>Replace a <code>ContentPlaceholder</code> object with a Paragraph object.</li> <li>Add to a <code>TextBoxPlaceholder</code> object a character vector, Paragraph object, or with a cell array of character vectors or Paragraph objects or a combination of both.</li> <li>Replace a <code>TextBoxPlaceholder</code> object with a Paragraph object.</li> </ul> See “Add and Replace Text in Placeholders” on page 14-60.

Text Specification Technique	Associated PPT API Objects
Add to, or replace, a text box.	Add to or replace a TextBox object with a character vector, Paragraph object, or with a cell array of character vectors or Paragraph objects, or a combination of both.  See "Add or Replace Text in a Text Box" on page 14-60.

### Add and Replace Text in Placeholders

You can add or replace text in a ContentPlaceholder and a TextBoxPlaceholder, specifying:

- A character vector
- A Paragraph object
- A cell array of character vectors or Paragraph objects or a combination of character vectors and Paragraph objects. An inner cell array specifies inner list (indented) items.

The slide layout specifies whether the text appears as paragraphs, a bulleted list, or a numbered list.

```
import mlreportgen.ppt.*

name1 = 'before';
ppt = Presentation(name1);
open(ppt);

add(ppt, 'Comparison');
replace(ppt, 'Left Content', 'dummy content');
replace(ppt, 'Right Content', 'dummy content');
close(ppt);

name2 = 'after';
slides = Presentation(name2, name1);

lefts = find(ppt, 'Left Content');
rights = find(ppt, 'Right Content');

para = replace(lefts(1), 'Left item in the list' );
para.Italic = true;
para.FontColor = 'green';

replace(rights(1), { ...
    'Right List item', ...
    { 'Inner right list item', 'Other inner right list item' }...
    'Right List item', ...
});

close(ppt);
rptview(ppt);
```

### Add or Replace Text in a Text Box

A text box in a slide is a box that you can add text to. You can programmatically add or replace the content of a text box in a presentation.

- 1 Create a TextBox object. Specify the location and width of the text box.
- 2 Add text by using the add method with the TextBox object.

- 3 Add the `TextBox` object to a presentation using the `add` method with a `Presentation` object or the `add` method with a `Slide` object.

For example:

```
import mlreportgen.ppt.*
ppt = Presentation('myPresentation.pptx');
open(ppt);
titleSlide = add(ppt, 'Title Slide');

tb = TextBox();
tb.X = '2in';
tb.Y = '2in';
tb.Width = '5in';
add(tb, 'Text for text box');

add(titleSlide, tb);
close(ppt);
```

## Add or Replace a Table

To add or replace a table in a presentation, use one these approaches:

- Add a table directly to a slide.
- Replace a placeholder from a slide layout with a table. For example, add a slide with a `Title` and `Content` or `Title` and `Table` layout and replace the content or table placeholder with a table.
- Replace a template table from a template presentation with a different table.

### Add Table to Blank Slide

Create an `mlreportgen.ppt.Table` object and add it to slide.

```
import mlreportgen.ppt.*
ppt = Presentation('myPresentation.pptx');
open(ppt);
tableSlide = add(ppt, 'Blank');
magicTable = Table(magic(5));
magicTable.X = '3in';
magicTable.Y = '5in';
add(tableSlide, magicTable);
close(ppt);
```

You can replace a table that you have already added to a slide by using the `replace` method. For example:

```
import mlreportgen.ppt.*
ppt = Presentation('myPresentation.pptx');
open(ppt);
tableSlide = add(ppt, 'Blank');
magicTable = Table(magic(5));
add(tableSlide, magicTable);
newTable = Table(magic(4));
replace(magicTable, newTable);
close(ppt);
```

### Replace Table Placeholder

You can replace a table placeholder that comes from a slide layout. For example, add a slide with a Title and Table layout. A table placeholder is represented by an `mreportgen.ppt.TablePlaceholder` object. To replace the table placeholder, use the `replace` method of the `TablePlaceholder` object.

```
import mreportgen.ppt.*
ppt = Presentation('myPresentation.pptx');
open(ppt);
tableSlide = add(ppt, 'Title and Table');
table1 = Table(magic(9));
tblplaceholderObj = find(tableSlide, 'Table');
replace(tblplaceholderObj, table1);
close(ppt);
```

### Replace Template Table

If you create a presentation from an existing presentation, a table from the existing presentation (a template table) is represented by an `mreportgen.ppt.TemplateTable` object. You can change the position, width, and height of the template table by setting properties of the object. You can also modify the XML markup of the template table. To replace the template table, use the `replace` method of the `TemplateTable` object. For example, suppose that you create a presentation from an existing presentation `myPresentation` that has a slide with the 'Title and Table' layout. The following code replaces the template table with another table.

```
import mreportgen.ppt.*
ppt = Presentation('myNewPresentation.pptx', 'myPresentation.pptx');
open(ppt);
slide1 = ppt.Children(1);
templateTableObj = find(slide1, 'Table');
replace(templateTableObj, Table(magic(4)));
close(ppt);
```

### Add or Replace a Picture

To add or replace a picture in a presentation, use one of these approaches:

- Add a picture directly to a slide.
- Replace a placeholder that comes from a slide layout with a picture. For example, add a slide with a Title and Content or Title and Picture layout and replace the content or picture placeholder with a picture.
- Replace a template picture from a template presentation with a different picture.

### Add Picture to Blank Slide

Create an `mreportgen.ppt.Picture` object and add it to slide.

```
import mreportgen.ppt.*
ppt = Presentation('myPresentation.pptx');
open(ppt);
pictureSlide = add(ppt, 'Blank');
plane = Picture(which('b747.jpg'));
plane.X = '2in';
plane.Y = '2in';
```

```
plane.Width = '5in';
plane.Height = '2in';
add(pictureSlide,plane);
close(ppt);
```

You can replace a picture that you have already added to a slide by using the `replace` method. For example:

```
import mlreportgen.ppt.*
ppt = Presentation('myPresentation.pptx');
open(ppt);
pictureSlide = add(ppt,'Blank');
plane = Picture(which('b747.jpg'));
plane.X = '2in';
plane.Y = '2in';
add(pictureSlide,plane);
peppers = Picture(which('peppers.png'));
replace(plane,peppers);
close(ppt);
```

### Replace Placeholder

You can replace a picture placeholder with a picture. For example, add a slide with a `Title` and `Picture` layout. A picture placeholder is represented by an `mlreportgen.ppt.PicturePlaceholder` object. To replace the picture placeholder, use the `replace` method of the `PicturePlaceholder` object.

```
import mlreportgen.ppt.*
ppt = Presentation('myPresentation.pptx');
open(ppt);
tableSlide = add(ppt,'Title and Picture');
plane = Picture(which('b747.jpg'));
plane.X = '2in';
plane.Y = '2in';
picplaceholderObj = find(tableSlide,'Picture');
replace(picplaceholderObj,plane);
close(ppt);
```

PowerPoint adjusts the picture dimensions to fit in the picture placeholder. If the picture placeholder dimensions are bigger than the `Picture` object dimensions, the picture stretches proportionally. If the dimensions are smaller, the picture is centered.

### Replace Template Picture

If you create a presentation from an existing presentation, a picture from the existing presentation (a template picture) is represented by an `mlreportgen.ppt.TemplatePicture` object. You can change the position, width, and height of the template picture by setting properties of the object. You can also modify the XML markup of the template picture. To replace the template picture, use the `replace` method of the `TemplatePicture` object. For example, suppose that you create a presentation from an existing presentation `myPresentation` that has a slide with the `'Title and Picture'` layout. The following code replaces the template picture with a different picture.

```
import mlreportgen.ppt.*
ppt = Presentation('myNewPresentation.pptx','myPresentation.pptx');
open(ppt);
slide1 = ppt.Children(1);
templateTableObj = find(slide1,'Picture');
```

```
replace(templateTableObj,Picture(which('peppers.png')));  
close(ppt);
```

### See Also

### Related Examples

- “Create and Format Text” on page 14-65
- “Create and Format Paragraphs” on page 14-67
- “Create and Format Tables” on page 14-69
- “Create and Format Pictures” on page 14-76
- “Create and Format Links” on page 14-78

### More About

- “Presentation Formatting Approaches” on page 14-18



## Create and Format Text

### In this section...

“Create Text” on page 14-65

“Create a Subscript or Superscript” on page 14-65

“Format Text” on page 14-65

### Create Text

You can create a `Text` object using an `mlreportgen.ppt.Text` constructor, specifying a character vector.

Also, you can create text by using a character vector or string scalar with objects of these PPT API classes:

- `Paragraph`
- `InternalLink`
- `ExternalLink`
- `TableEntry`
- `TextBox`
- `ContentPlaceholder`
- `TextBoxPlaceholder`

For example:

```
import mlreportgen.ppt.*;
ppt = Presentation('myPresentation.pptx');
slide1 = add(ppt, 'Title Slide');

contents = find(slide1, 'Title');
titleText = replace(contents(1), 'My Title');
```

For more information about creating and adding text, see “Add and Replace Text” on page 14-59.

### Create a Subscript or Superscript

You can enable the `Subscript` or `Superscript` property for a `Text` object. Enabling these properties specifies that the text gets treated as a subscript or superscript when you add it to a `Paragraph` object. For example, you can set up a paragraph to display  $x^2$ .

```
super = Text('2');
super.Superscript = true;

para = Paragraph('x');
append(para, super);
```

### Format Text

To format a `Text` object, use format objects with a `Text` object `Style` property or use `Text` object properties. For example:

```
t = Text('green text');  
t.Style = {Bold(true)};  
t.FontColor = 'green';
```

Text Object Formatting	Format Object	Format Property
Font family	FontFamily	Font
Font family for complex scripts to handle locales	FontFamily	ComplexScriptFont
Font size	FontSize	FontSize
Font color	FontColor	FontColor
Bold	Bold	Bold
Italic	Italic	Italic
Strike	Strike	Strike
Underline	Underline	Underline
Subscript	Subscript	Subscript
Superscript	Superscript	Superscript

## See Also

### Classes

`mlreportgen.ppt.Text` | `mlreportgen.ppt.Paragraph` | `mlreportgen.ppt.InternalLink` | `mlreportgen.ppt.ExternalLink` | `mlreportgen.ppt.TextBox`

## Related Examples

- “Add and Replace Text” on page 14-59

## More About

- “Presentation Formatting Approaches” on page 14-18

## Create and Format Paragraphs

### In this section...

“Create a Paragraph” on page 14-67

“Format Paragraph Content” on page 14-67

### Create a Paragraph

To create a Paragraph object, use the `mlreportgen.ppt.Paragraph` constructor. You can:

- Create an empty Paragraph object.
- Specify a character vector or string scalar for the paragraph text.
- Specify a Text, InternalLink, or ExternalLink object as the paragraph text.

After you create a Paragraph object, you can append character vectors, string scalars, or Text, InternalLink, or ExternalLink objects to add text to the paragraph. You can specify separate formatting for each object that you append.

### Format Paragraph Content

You can specify the default formatting to apply to the text in a paragraph. The paragraph formatting applies to text that you add. The paragraph formatting applies to Text, InternalLink, and ExternalLink objects in the paragraph, unless those objects specify formatting that overrides the default paragraph formatting. For example, this code produces alternating red and green text:

```
p = Paragraph('Default paragraph green text');
p.FontColor = 'green';

redText = Text(' red text');
redText.FontColor = 'red';
append(p, redText);

moreText = Text(' back to default green text');
append(p, moreText);
```

- Default paragraph green text red text back to default green text

Paragraph Object Formatting	Format Object	Format Property
Font family	FontFamily	Font
Font family for complex scripts to handle locales	FontFamily	ComplexScriptFont
Font size	FontSize	FontSize
Bold	Bold	Bold
Font color	FontColor	FontColor
Italic	Italic	Italic
Strike	Strike	Strike

<b>Paragraph Object Formatting</b>	<b>Format Object</b>	<b>Format Property</b>
Underline	Underline	Underline
Subscript	Subscript	Subscript
Superscript	Superscript	Superscript
Horizontal alignment	HAlign	HAlign
Level of indentation Use the PowerPoint template to specify formatting for each level.	n/a	Level

---

**Tip** Although you can specify that text in a Paragraph object is a subscript or superscript, using Text objects with Subscript or Superscript property set gives you greater formatting flexibility.

---

## See Also

### Classes

`mreportgen.ppt.Text` | `mreportgen.ppt.Paragraph` | `mreportgen.ppt.ExternalLink` | `mreportgen.ppt.InternalLink` | `mreportgen.ppt.TextBox`

### Classes

`mreportgen.ppt.TableEntry`

## Related Examples

- “Add and Replace Text” on page 14-59

## More About

- “Presentation Formatting Approaches” on page 14-18

## Create and Format Tables

### Create a Table

To create a table, you can:

- Create an empty `Table` object using the `mlreportgen.ppt.Table` constructor without arguments. Then append `TableRow` objects to the `Table` object and append `TableEntry` objects to the `TableRow` objects.
- Create an empty `Table` object using the `mlreportgen.ppt.Table` constructor, specifying the number of columns.
- Create a `Table` object whose rows and columns are populated by the values you specify in the constructor. You can specify a two-dimensional numeric array or a two-dimensional cell array of numbers, character vectors, and `Paragraph` objects. You can also use a combination of these kinds of values.

For an example of creating a table by appending table rows to an empty table, see `mlreportgen.ppt.TableRow`. For an example of creating a table by specifying values in the `Table` object constructor, see `mlreportgen.ppt.Table`.

### Format a Table

You can specify a table style name for the overall look of a table, such as a table that shades alternating rows. You can set the `StyleName` property of a `Table` object to the name of a table style.

#### Table Styles in Templates

The PowerPoint template must contain an instance of the table style for you to use it in a PPT API program. To list the instances of table styles in your template, use `getTableStyleNames`.

```
import mlreportgen.ppt.*

%% Create a new presentation and open it
slides = Presentation('myPrsentation');
open(ppt);

%% Print out all table styles and
%% their universally unique identifiers (UUID)
pptStyles = getTableStyleNames(slides);
fprintf('Available table styles:\n');
for i = 1:length(pptStyles)
    fprintf('    Style name: '%s'\n', pptStyles{i,1});
    fprintf('    UUID: '%s'\n', pptStyles{i,2});
end

%% Close the presentation
close(ppt);
```

Each style returned has a name and an ID. You can use the name or the ID with the `Style` property. Use the ID when the name can vary based on locale.

```
Available table styles:
    Style name: 'Medium Style 2 - Accent 1'
```

```

        UUID: '{5C22544A-7EE6-4342-B048-85BDC9FD1C3A}'
    Style name: 'Light Style 1'
        UUID: '{9D7B26C5-4107-4FEC-AEDC-1716B250A1EF}'
    Style name: 'Light Style 1 - Accent 1'
        UUID: '{3B4B98B0-60AC-42C2-AFA5-B58CD77FA1E5}'
    Style name: 'Light Style 1 - Accent 2'
        UUID: '{0E3FDE45-AF77-4B5C-9715-49D594BDF05E}'

```

If the name of the style you want to use does not have an instance, create one.

- 1 Create a slide in your PowerPoint template.
- 2 In the slide, create a table.
- 3 Apply the styles that you want to use in your program to the table. Applying a style creates an instance of the style in the template.
- 4 Delete the slide, and save and close the template.

### Format a Table Using a Table Style

This example shows how to format a table using a table style.

```

import mlreportgen.ppt.*

%% Create a new presentation and add two slides to it
ppt = Presentation();
open(ppt);

add(ppt,'Title and Content');
add(ppt,'Title and Content');

%% Save the two content placeholders named 'Content' in an array.
%% Replace the first content placeholder with a 5x5 table and
%% apply a table style to it.
contents = find(ppt,'Content');
tbl = replace(contents(1),Table(magic(5)));
tbl.StyleName = 'Medium Style 2 - Accent 1'

%% Replace the second content placeholder with a 10x10 table and
%% apply a different table style.
%% Generate the presentation and open it.
tbl = replace(contents(2),Table(magic(10)));
tbl.StyleName = 'Medium Style 2 - Accent 2'
close(ppt);
rptview(ppt);

```

This code creates a PowerPoint presentation that has two slides. Each slide contains a table, and each table has a different table style applied to it.

### Formatting Options

You can specify the location (upper-left x and y coordinates), height, and width properties of a table. When you add the table to a presentation programmatically, PowerPoint uses those properties, if all of the table content fits in the table. When you replace a `TablePlaceholder` or `ContentPlaceholder` with a table, PowerPoint fits the table in the placeholder location and dimensions.

You can specify default formatting for the contents of a table, a column, a table row, and a table entry. Table entry formatting takes precedence over the formatting you specify for a column or for a table row. Table row formatting takes precedence over table formatting.

You can specify these default formatting options for the contents of a `Table` object.

<b>Table Object Formatting</b>	<b>Format Object</b>	<b>Format Property</b>
Table style from template  Use the PowerPoint template to specify table style formatting. Create an instance of the style in your template.	n/a	StyleName
Background color	BackgroundColor	BackgroundColor
Column formatting	ColSpec	ColSpecs
Vertical alignment of table cell content	VAlign	VAlign
Font family	FontFamily	Font
Font family for complex scripts to handle locales	FontFamily	ComplexScriptFont
Font size	FontSize	FontSize
Font color	FontColor	FontColor
Upper-left x-coordinate of the table	n/a	X
Upper-left y-coordinate of the table	n/a	Y
Table width	n/a	Width
Table height	n/a	Height

To specify default formatting for the contents of a `TableRow` object, use the `Style` property with these format objects.

<b>TableRow Object Formatting</b>	<b>Format Object</b>	<b>Format Property</b>
Background color	BackgroundColor	n/a
Vertical alignment of table cell content	VAlign	n/a
Font family	FontColor	n/a
Font family for complex scripts	FontFamily	n/a
Font size	FontSize	n/a
Text color	FontColor	n/a
Bold	Bold	n/a
Italic	Italic	n/a
Strike	Strike	n/a
Underline	Underline	n/a

<b>TableRow Object Formatting</b>	<b>Format Object</b>	<b>Format Property</b>
Background color	BackgroundColor	n/a

To specify default formatting for the contents of a `TableEntry` object, use these formatting options.

<b>TableEntry Object Formatting</b>	<b>Format Object</b>	<b>Format Property</b>
Background color	BackgroundColor	BackgroundColor
Column width	ColWidth	n/a
Vertical alignment of table cell content	VAlign	VAlign
Font family	FontFamily	Font
Font family for complex scripts to handle locales	FontFamily	ComplexScriptFont
Text color	FontColor	FontColor
Font size	FontSize	FontSize
Bold	Bold	n/a
Italic	Italic	n/a
Strike	Strike	n/a
Underline	Underline	n/a

### Access a Table Row or Entry

To access a row in a table, use the `mlreportgen.ppt.Table.row` method. Specify the `Table` object and the number of the row you want to access. For example, to access a `TableRow` object for the second row of `myTable`, use:

```
myTable = Table(magic(5));
row2 = row(myTable,2);
```

To access an entry in a table, use the `mlreportgen.ppt.Table.entry` method. Specify the `Table` object and the number of the row and the number of the column that you want to access. For example, to access a `TableEntry` object for the third entry in the second row of `myTable`, use:

```
myTable = Table(magic(5));
entry3row2 = entry(myTable,2,3);
```

Alternatively, you can access a table row by using the `Children` property of a `Table` object. You can access a table entry by using the `Children` property of a `TableRow` object. For example, to access the third entry in the second row of `myTable`:

```
myTable = Table(magic(5));
entry3row2 = myTable.Children(2).Children(3);
```

### Format a Column

To format a column in a table, use one or more `mlreportgen.ppt.ColSpec` objects. Create a `ColSpec` object for each column that you want to format and specify the formatting for each `ColSpec` object. Then define an array of the `ColSpec` objects and use that with the `ColSpecs` property of the `Table` object.



The format specification for a table row takes precedence over the format specification for a column.

```
import mlreportgen.ppt.*
ppt = Presentation('myColSpecs.pptx');
open(ppt);

add(ppt, 'Title and Content');

t = Table(magic(12));
t.Style = {HAlign('center')};

colSpecs(2) = ColSpec('1.5in');
colSpecs(1) = ColSpec('1.5in');
colSpecs(1).BackgroundColor = 'red';
colSpecs(2).BackgroundColor = 'green';
t.ColSpecs = colSpecs;
t.row(2).Style = {VAlign('bottom')};
t.row(2).BackgroundColor = 'tan';
t.entry(2,3).FontColor = 'red';
t.entry(2,3).FontSize = '30pt';

replace(ppt, 'Content', t);

close(ppt);
rptview(ppt);
```

When you create a ColSpec object, you can specify the column width in the constructor. For example:

```
myColSpec = ColSpec('3in');
```

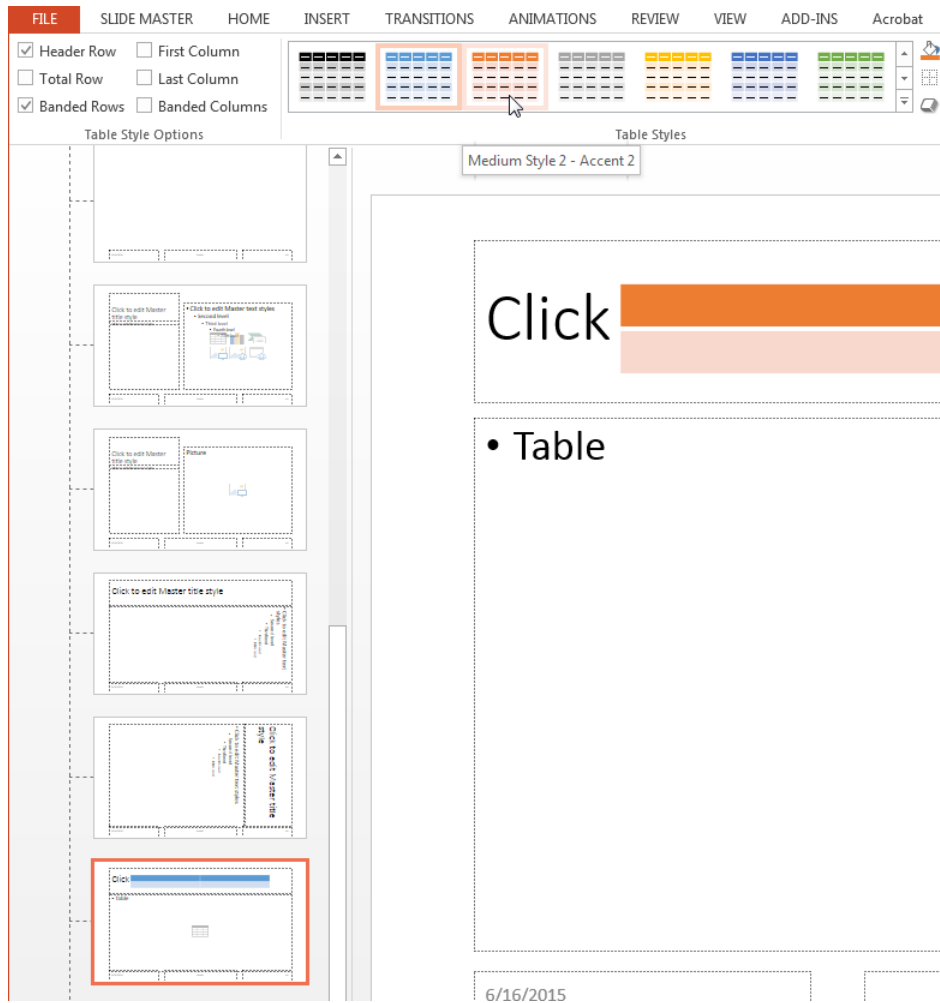
Also, you can specify the column width using the Width property of a ColSpec object. You specify other formatting properties of the ColSpec object, such as BackgroundColor.

## View Table Style Names

If you use the PPT API, to specify a table style other than the default, you need to know the names of table styles in a PowerPoint template. You can view the name in PowerPoint or using the PPT API.

- 1 In PowerPoint, select **View > Slide Master**.
- 2 In a slide layout that has a table, click **Table** (or anywhere in that placeholder). On the **Insert** tab, click **Table**.
- 3 Create an empty table in the slide layout.

A panel of **Table Styles** appears. To see the name of a table style, hover over the table style image.



To see table style names using the PPT API, use the `getTableStyleNames` method with an `m1reportgen.ppt.Presentation` object. The output in this example shows just the first two of many table styles in the default template.

```
import m1reportgen.ppt.*
ppt = Presentation('myPlaceholderPresentation');

getTableStyleNames(ppt)

ans =

'Medium Style 2 - Accent 1'      '{5C22544A-7EE6-4342-B048-85BDC9FD1C3A}'
'Light Style 1'                 '{9D7B26C5-4107-4FEC-AEDC-1716B250A1EF}'
```

To use a table style name with the PPT API, you can use either the name or the numeric identifier.

## See Also

### Functions

`row | entry`

**Classes**

mlreportgen.ppt.Table | mlreportgen.ppt.TableRow | mlreportgen.ppt.TableEntry |  
mlreportgen.ppt.TablePlaceholder | mlreportgen.ppt.ColSpec |  
mlreportgen.ppt.ColWidth

**Related Examples**

- “Add or Replace a Table” on page 14-61

**More About**

- “Presentation Formatting Approaches” on page 14-18

## Create and Format Pictures

### In this section...

“Create a Picture” on page 14-76

“Format a Picture” on page 14-76

### Create a Picture

To create a picture for a presentation, use the `m1reportgen.ppt.Picture` constructor. Specify the path to a picture file. For example:

```
import m1reportgen.ppt.*
ppt = Presentation('slides');
pictureSlide = add(ppt, 'Blank');

plane = Picture(which('b747.jpg'));
plane.Width = '5in';
plane.Height = '2in';

add(pictureSlide, plane);

close(ppt);
```

The PPT API supports the image formats in the table.

Image Format	File Extension
Bitmap	.bmp
Enhanced Metafile	.emf
Encapsulated PostScript	.eps
Graphics Interchange Format	.gif
Joint Photographic Experts Group	.jpeg, .jpg
Portable Network Graphics	.png
Scalable Vector Graphics	.svg
Tagged Image File Format	.tif, .tiff

**Note** To ensure that the image formats used in your generated presentation are supported by the PowerPoint version that you are using, see File formats that are supported in PowerPoint on the Microsoft website.

### Format a Picture

When you create a `Picture` object, you can specify the location, width, and height. The specified formatting applies when you add a picture to a slide or replace a `Picture` object. When you replace a `PicturePlaceholder` object with a `Picture` object, PowerPoint adjusts the replacement picture to fit the location and dimensions of the placeholder.

You can specify these format properties for a `Picture` object.

Picture Object Formatting	Format Object	Format Property
Upper-left x-coordinate of picture	n/a	X
Upper-left y-coordinate of picture	n/a	Y
Picture width	n/a	Width
Picture height	n/a	Height

## See Also

### Classes

`mlreportgen.ppt.Picture` | `mlreportgen.ppt.PicturePlaceholder`

## Related Examples

- “Add or Replace a Picture” on page 14-62

## More About

- “Presentation Formatting Approaches” on page 14-18

## Create and Format Links

In a presentation, you can create an internal link, from one slide to another slide, or an external link to a location outside of the presentation.

### Create an Internal Link

To create a link from one slide to another slide, append an object of the `mlreportgen.ppt.InternalLink` class to a paragraph in the source slide. Use the `InternalLink` object properties to specify the target slide and the link text. You can specify the target slide by using the target slide name or index.

#### Link to Slide Using the Target Slide Name

This example links to a slide using an `mlreportgen.ppt.InternalLink` object that specifies the target slide name.

Create the presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myPresentation1.pptx");
open(ppt);
```

Add a slide to the presentation.

```
slide1 = add(ppt,"Title and Content");
```

Choose a name to identify the target slide.

```
targetSlideName = "myTargetSlide";
```

Create a paragraph. Create an `InternalLink` object that specifies the target slide by name and append it to the paragraph.

```
p = Paragraph("This is a link to the slide with the name ");
linkObj = InternalLink(targetSlideName,targetSlideName);
append(p,linkObj);
```

Add the title and content to the slide.

```
replace(slide1,"Title","First slide");
replace(slide1,"Content",p);
```

Add a second slide to the presentation.

```
slide2 = add(ppt,"Title and Content");
replace(slide2,"Title","Second slide");
```

Add the target slide to the presentation. Set the `Name` property of the slide to the name specified in the `InternalLink` object.

```
slide3 = add(ppt,"Title and Content");
slide3.Name = targetSlideName;
replace(slide3,"Title","Third slide");
content = strcat("This is the target slide with the name ",targetSlideName);
replace(slide3,"Content",content);
```

Close and view the presentation.

```
close(ppt);  
rptview(ppt);
```

Here are the generated slides:

1

First slide

- This is a link to the slide with the name [myTargetSlide](#)

2

Second slide

3

Third slide

- This is the target slide with the name myTargetSlide

### Link to a Slide Using the Target Slide Index

This example links to a slide using an `mlreportgen.ppt.InternalLink` object that specifies the target slide number.

Create the presentation.

```
import mlreportgen.ppt.*
ppt = Presentation("myPresentation2.pptx");
open(ppt);
```

Add a slide to the presentation.

```
slide1 = add(ppt, "Title and Content");
```

Create a paragraph. Create an `InternalLink` object that specifies the target slide by its index and append the object to the paragraph.

```
p = Paragraph("This is a link to ");
link = InternalLink(3, "slide 3");
append(p, link);
```

Add the title and content to the slide.

```
replace(slide1, "Title", "First slide");
replace(slide1, "Content", p);
```

Add a slide 2 to the presentation.

```
slide2 = add(ppt, "Title and Content");
replace(slide2, "Title", "Second slide");
```

Add the target slide, slide 3, to the presentation.

```
slide3 = add(ppt, "Title and Content");
replace(slide3, "Title", "Third slide");
replace(slide3, "Content", "This is the target slide");
```

Close and view the presentation.

```
close(ppt);
rptview(ppt);
```

Here are the generated slides:



1

First slide

- This is a link to [slide 3](#)

8/25/2020 1

2

Second slide

8/25/2020 2

3

Third slide

- This is the target slide

8/25/2020 3

## Create an External Link

To create a link to a location outside of a presentation, append an object of the `mlreportgen.ppt.ExternalLink` class to a paragraph in a slide. Use the `ExternalLink` object properties to specify the link text, and the full URL of the link target .

### Link from a Slide to a Website

This example uses an `mlreportgen.ppt.ExternalLink` object to link from a slide to a website.

```
import mlreportgen.ppt.*

ppt = Presentation("myPresentation3.pptx");
open(ppt);

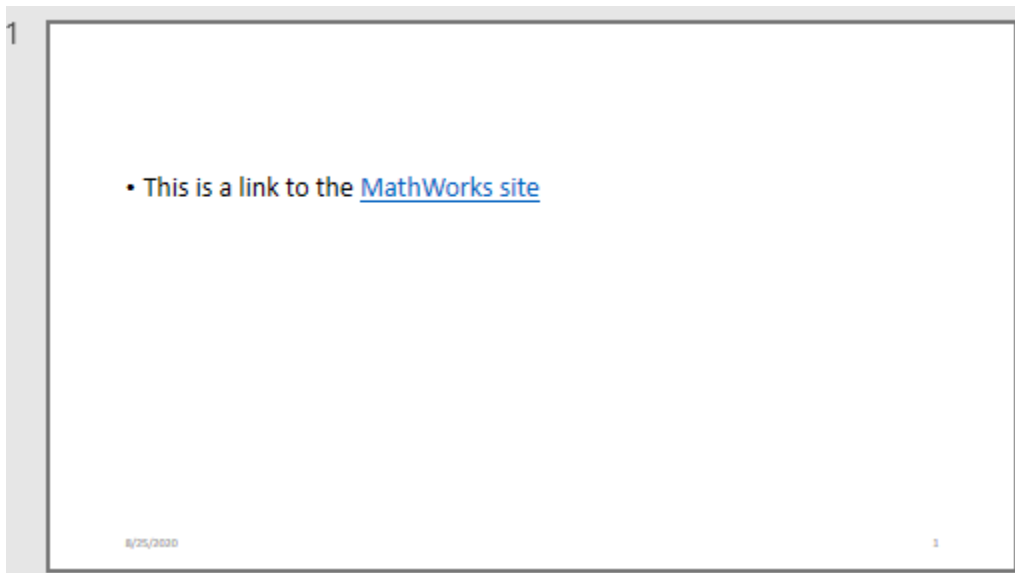
add(ppt, "Title and Content");

p = Paragraph("This is a link to the ");
link = ExternalLink("https://www.mathworks.com", "MathWorks site");

append(p, link);
replace(ppt, "Content", p);

close(ppt);
rptview(ppt);
```

Here is the generated slide:



## Format an Internal or External Link

You can customize the appearance of the link text by using the format properties of an `mlreportgen.ppt.InternalLink` or `mlreportgen.ppt.ExternalLink` object or by adding format objects to the `Style` property of an `InternalLink` or `ExternalLink` object. See

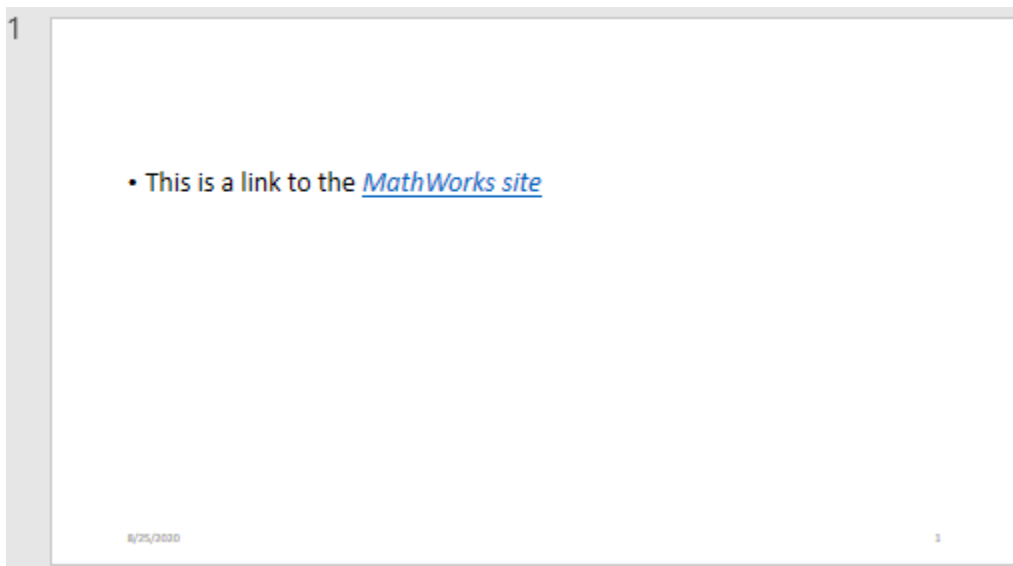
`mlreportgen.ppt.InternalLink` and `mlreportgen.ppt.ExternalLink` for the properties that specify the link text appearance.

### Customize the Appearance of the Link Text

This example specifies italic link text for a link from a slide to a website.

```
import mlreportgen.ppt.*
ppt = Presentation("myPresentation4.pptx");
open(ppt);
add(ppt,"Title and Content");
p = Paragraph("This is a link to the ");
link = ExternalLink("https://www.mathworks.com","MathWorks site");
link.Italic = true;
append(p,link);
replace(ppt,"Content",p);
close(ppt);
rptview(ppt);
```

Here is the generated slide:



## See Also

### Classes

`mlreportgen.ppt.ExternalLink` | `mlreportgen.ppt.InternalLink`

## More About

- “Presentation Formatting Approaches” on page 14-18

## **Generate a Presentation From the Results of a MATLAB Application**

This example shows how to use the MATLAB® API for PowerPoint® (PPT API) to generate a Microsoft® PowerPoint® presentation from the results of a MATLAB application. The example generates a presentation from the results of an application that predicts the United States population. Here are the slides that the example generates:

1

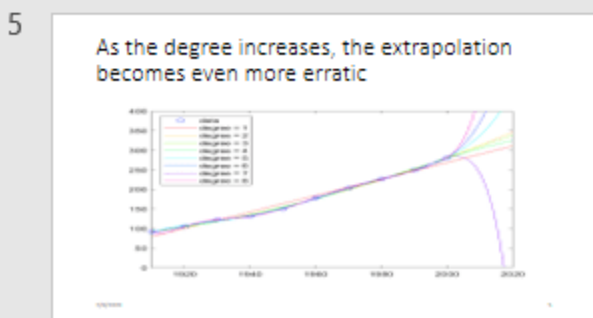
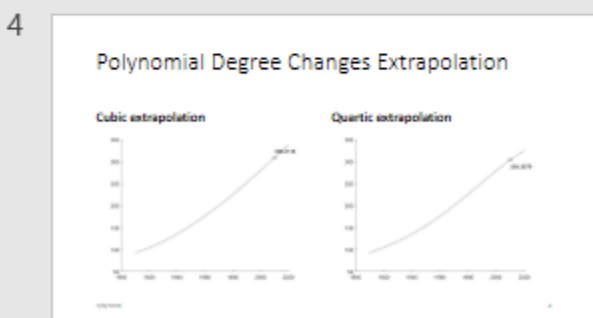
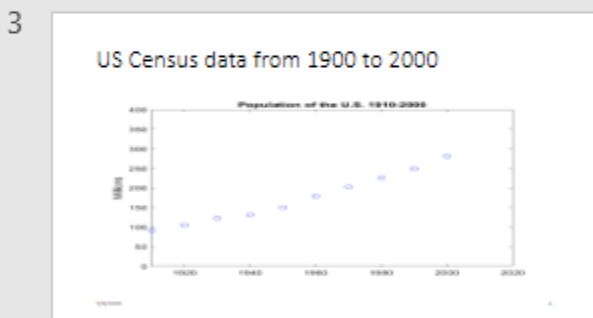
## Modeling the US Population

A Risky Business

2

### Population Modeling Approach

- Fit polynomial to U.S. Census data
- Use polynomials to extrapolate population growth
- Based on "Computer Methods for Mathematical Computations", by Forsythe, Malcolm and Moler, published by Prentice-Hall in 1977
- Varying polynomial degree shows riskiness of approach



### Create the Presentation

Import the PPT package so that you do not have to use long, fully qualified names.

```
import mlreportgen.ppt.*;
```

To facilitate deleting the images that the example generates, create a cell array to hold the images.

```
images = {};
```

Create a presentation, using the default template.

```
ppt = Presentation('population.pptx');  
open(ppt);
```

### Add Slides to the Presentation

PowerPoint presentations consist of slides that are created from predefined layouts. The layouts contain placeholders that you fill with generated content. The predefined layouts belong to a template slide master that defines the styles.

Add the first slide to the presentation, using the `Title Slide` layout.

```
slide1 = add(ppt, 'Title Slide');
```

Replace the title and subtitle in the slide by using the `replace` method.

```
replace(slide1, 'Title', 'Modeling the US Population');  
replace(slide1, 'Subtitle', 'A Risky Business');
```

Add the second slide to the presentation using the `Title and Content` layout. Replace the title.

```
slide2 = add(ppt, 'Title and Content');  
replace(slide2, 'Title', 'Population Modeling Approach');
```

Add text to the `Content` placeholder using a cell array.

```
replace(slide2, 'Content', { ...  
    'Fit polynomial to U.S. Census data' ...  
    'Use polynomials to extrapolate population growth' ...  
    ['Based on "Computer Methods for Mathematical Computations",' ...  
    ' by Forsythe, Malcolm and Moler, published by Prentice-Hall in 1977'] ...  
    'Varying polynomial degree shows riskiness of approach'});
```

Add the third slide to the presentation using the `Title and Content` layout. Replace the title.

```
slide3 = add(ppt, 'Title and Content');  
replace(slide3, 'Title', 'US Census data from 1900 to 2000');
```

Create a plot of US census data from 1910 to 2000.

```
% Time interval  
t = (1910:10:2000)';  
  
% Population  
p = [91.972 105.711 123.203 131.669 150.697...  
    179.323 203.212 226.505 249.633 281.422]';  
  
% Plot
```

```

fig1 = figure;
plot(t,p,'bo');
axis([1910 2020 0 400]);
title('Population of the U.S. 1910-2000');
ylabel('Millions');

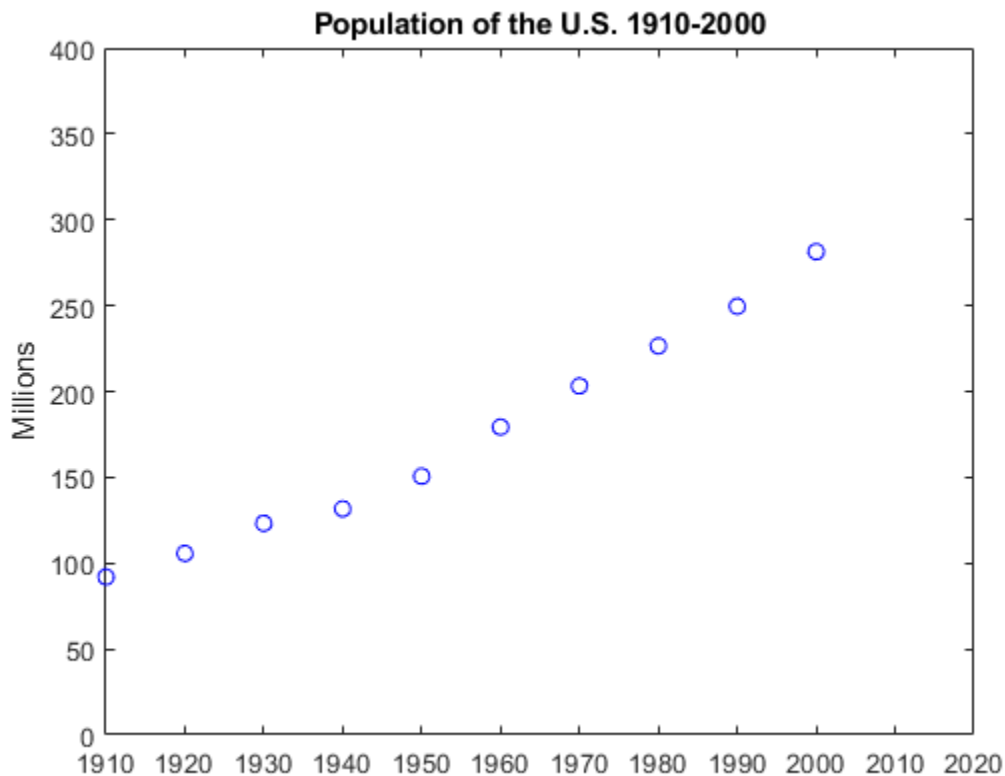
```

Convert the plot to an image. Add the image to the list of images to be deleted at the end of presentation generation. You must not delete the images until after you close the presentation.

```

img1 = 'plot1.png';
saveas(fig1,img1);

```



```

images = [images {img1}];

```

Replace the Content placeholder with the image.

```

replace(slide3, 'Content', Picture(img1));

```

Add the fourth slide to the presentation using the Comparison layout. Use this slide to show a comparison of the cubic and quartic extrapolations of the population data.

```

slide4 = add(ppt, 'Comparison');
replace(slide4, 'Title', 'Polynomial Degree Changes Extrapolation');

```

Compute the coefficients for a polynomial approximation of the population data.

```

n = length(t);
s = (t-1950)/50;

```

```
A = zeros(n);
A(:,end) = 1;
for j = n-1:-1:1
    A(:,j) = s .* A(:,j+1);
end
c = A(:,n-3:n)\p;
```

Replace the Left Text placeholder with text.

```
replace(slide4, 'Left Text', 'Cubic extrapolation');
```

Compute the cubic extrapolation.

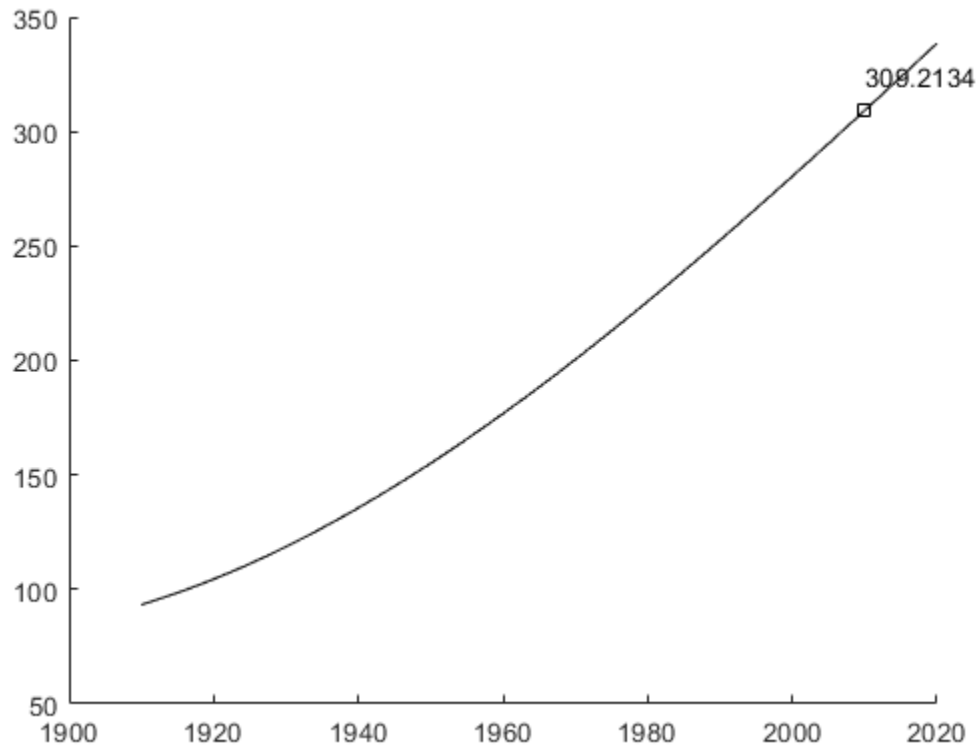
```
v = (1910:2020)';
x = (v-1950)/50;
w = (2010-1950)/50;
y = polyval(c,x);
z = polyval(c,w);
```

```
fig2 = figure;
hold on
plot(v,y,'k-');
plot(2010,z,'ks');
text(2010,z+15,num2str(z));
hold off
```

Create an image from the plot and add the image to the list of images to be deleted.

```
img2 = 'plot2.png';
saveas(fig2,img2);
```





```
images = [images {img2}];
```

Replace the Left Content placeholder with the image.

```
replace(slide4, 'Left Content', Picture(img2));
```

Replace the Right Text placeholder with text.

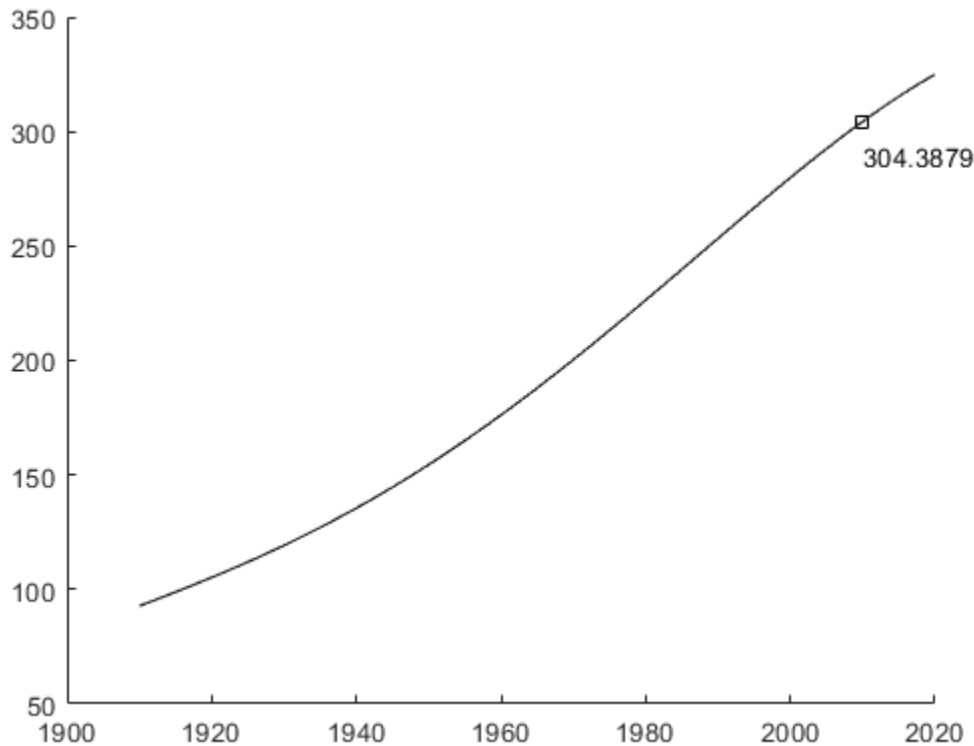
```
replace(slide4, 'Right Text', 'Quartic extrapolation');
```

Compute the quartic extrapolation.

```
c = A(:,n-4:n)\p;
y = polyval(c,x);
z = polyval(c,w);
fig3 = figure;
hold on
plot(v,y, 'k-');
plot(2010,z, 'ks');
text(2010,z-15,num2str(z));
hold off
```

Create an image from the plot, add the image to the list of images to be deleted, and replace the Right Content placeholder with the image.

```
img3 = 'plot3.png';
saveas(fig3,img3);
```



```
images = [images {img3}];
replace(slide4, 'Right Content', Picture(img3));
```

Add the last slide to the presentation using the Title and Content layout.

```
slide5 = add(ppt, 'Title and Content');
replace(slide5, 'Title', 'As the degree increases, the extrapolation becomes even more erratic');
```

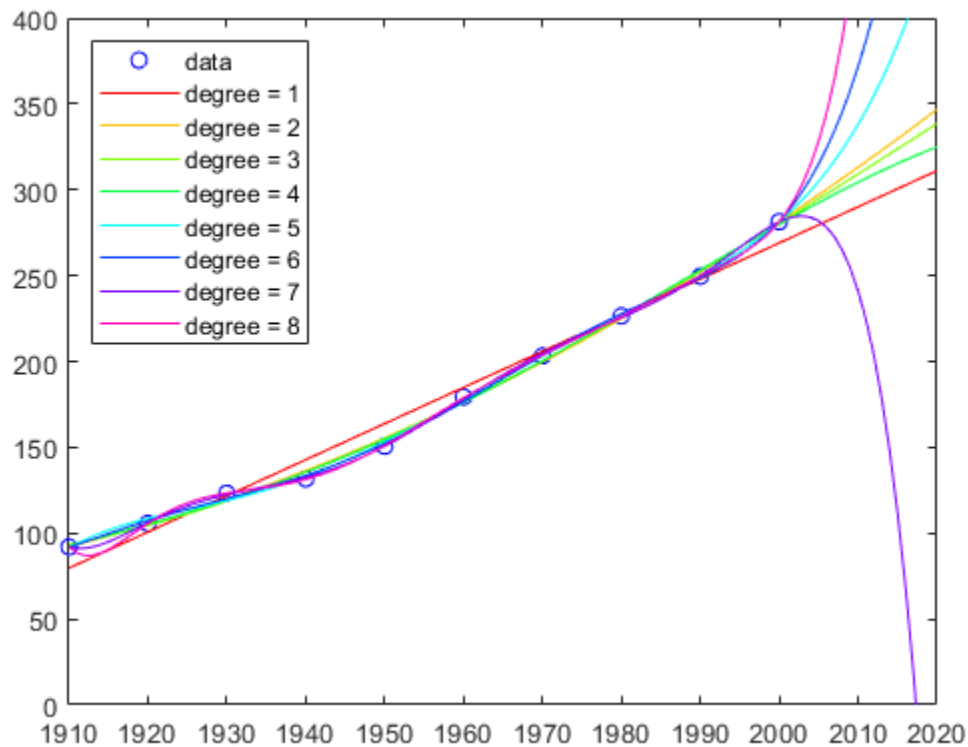
Create a plot to demonstrate that as the degree increases, the extrapolation becomes even more erratic.

```
fig4 = figure;
cla
plot(t,p, 'bo')
hold on
axis([1910 2020 0 400])
colors = hsv(8);
labels = {'data'};
for d = 1:8
    [Q,R] = qr(A(:,n-d:n));
    R = R(1:d+1,:);
    Q = Q(:,1:d+1);
    c = R\ (Q'*p);
    y = polyval(c,x);
    z = polyval(c,11);
    plot(v,y, 'color', colors(d,:));
    labels{end+1} = ['degree = ' int2str(d)];
end
```

```
legend(labels, 'Location', 'NorthWest')
hold off
```

Create an image from the plot and replace the Content placeholder with the image.

```
img4 = 'plot4.png';
saveas(fig4,img4);
```



```
images = [images {img4}];
replace(slide5, 'Content', Picture(img4));
```

### Close and View the Presentation

```
close(ppt);
rptview(ppt);
```

### Delete the Images

When the presentation closes, the images are copied into the presentation. Now, you can delete the images.

```
len = length(images);
for i = 1:len
    delete(images{i});
end
```

### See Also

mlreportgen.ppt.Presentation | mlreportgen.ppt.Slide

## **More About**

- “Create a Presentation Programmatically” on page 14-49
- “Predicting the US Population”

## **Classes Being Removed**

---

## mlreportgen.dom.DOCXRawFormat class

**Package:** mlreportgen.dom

XML markup for array of Microsoft Word formats

### Compatibility

---

**Note** mlreportgen.dom.DOCXRawFormat will be removed in a future release. Use mlreportgen.dom.PageRawFormat instead.

---

### Description

XML markup for an array of Microsoft Word formats.

### Construction

docxRawFormatObj = DOCXRawFormat() creates an empty array of raw formats.

### Output Arguments

**docxRawFormatObj** — XML markup for Word formats

mlreportgen.dom.DOCXRawFormat object

XML markup for Word formats, represented by an mlreportgen.dom.DOCXRawFormat object.

### Properties

**Id** — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Markup** — Word XML markup character vectors

cell array of character vectors

Specify a cell array of character vectors. Each character vector contains Word XML markup for a Word format.

For information about XML markup for Word formats, see <https://www.ecma-international.org/publications-and-standards/standards/ecma-376/>.

**Tag** — Tag for this document element

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the Id property of the

object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Examples

### Turn on Line Numbering Based on Default DOM Template

In this example, the `RawFormats` property of a `DOCXSection` is initialized with the markup for properties specified by the default template. This code appends the line numbering property to the existing properties.

```
import mlreportgen.dom.*;
d = Document('myreport', 'docx');
open(d);

s = d.CurrentDOCXSection;
s.RawFormats = [s.RawFormats ...
{'<w:lnNumType w:countBy="1" w:start="1" w:restart="newSection"/>' }];
append(d, 'This document has line numbers');

close(d);
rptview('myreport', 'docx');
```

### See Also

`mlreportgen.dom.DOCXSection`

### Topics

“Report Formatting Approaches” on page 13-17

## mlreportgen.dom.DOCXPageSize class

**Package:** mlreportgen.dom

Size and orientation of pages in Microsoft Word document

### Compatibility

---

**Note** mlreportgen.dom.DOCXPageSize will be removed in a future release. Use mlreportgen.dom.PageSize instead.

---

### Description

Specifies the height, width, and orientation of pages in a section of a Microsoft Word document.

### Construction

`docxPageSizeObj = DOCXPageSize()` creates a page size object having default values of 8.5-by-11 inches and portrait orientation.

`docxPageSizeObj = DOCXPageSize(height,width)` creates a portrait page having a specified height and width.

`docxPageSizeObj = DOCXPageSize(height,width,orientation)` creates a page having a specified height, width, and orientation.

### Input Arguments

#### **height** — Height of page

'11in' (default) | character vector

character vector specifying the height of the page. The character vector must have the format `valueUnits` where `Units` is an abbreviation for the units in which the width size is expressed. Valid abbreviations are:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pc` — picas
- `pt` — points
- `px` — pixels

#### **width** — Width of page

'8.5in' (default) | character vector

The character vector must have the format `valueUnits`, where `Units` is an abbreviation for the units in which the width size is expressed. Valid abbreviations are:



- no abbreviation — pixels
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points
- px — pixels

### **orientation — Orientation of page**

character vector

Use one of these values:

- 'portrait' (default)
- 'landscape'

Specify height and width values that reflect the orientation setting. For example, if the orientation is landscape and the document is to be printed on 8.5x11-inch paper, set `height` to '8.5in' and `width` to '11in'.

### **Output Arguments**

#### **docXPageSizeObj — Page size and orientation of Word document**

mlreportgen.dom.DOCXPageSize object

Page size and orientation of a Word document, represented by an `mlreportgen.dom.DOCXPageSize` object.

## **Properties**

### **Height — Height of pages in Word page layout section**

character vector

character vector specifying the page height. The character vector must have the format `valueUnits` where `Units` is an abbreviation for the units in which the width size is expressed. Valid abbreviations are:

- no abbreviation — pixels
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points
- px — pixels

### **Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Orientation — Orientation (portrait or landscape) for pages in section**

character vector

Use one of these values:

- 'portrait' (default)
- 'landscape'

Specify height and width values that reflect the orientation setting. For example, if the orientation is landscape and the document is to be printed on 8.5x11-inch paper, set `height` to '8.5in' and `width` to '11in'.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form CLASS:ID, where CLASS is the object class and ID is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Width — Width of page**

'8.5in' (default) | character vector

The character vector must have the format `valueUnits`, where `Units` is an abbreviation for the units in which the width size is expressed. Valid abbreviations are:

- no abbreviation — pixels
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points
- px — pixels

**Examples****Set Page Orientation and Size**

Change the page orientation and size specified by the default DOM template.

```
import mlreportgen.dom.*;
d = Document('myreport', 'docx');
open(d);

s = d.CurrentDOCXSection;
s.PageSize.Orientation = 'landscape';
s.PageSize.Height = '8.5in';
s.PageSize.Width = '11in';
append(d, 'This document has landscape pages');
```

```
close(d);  
rptview('myreport', 'docx');
```

## See Also

[mlreportgen.dom.DOCXSection](#) | [mlreportgen.dom.DOCXPageMargins](#)

## Topics

“Report Formatting Approaches” on page 13-17

## mlreportgen.dom.DOCXPageMargins class

**Package:** mlreportgen.dom

Page margins for Microsoft Word page layout

### Compatibility

---

**Note** mlreportgen.dom.DOCXPageMargins will be removed in a future release. Use mlreportgen.dom.PageMargins instead.

---

### Description

Specifies the size of the page margins of a section of a Microsoft Word document.

### Construction

docxPageMarginsObj = DOCXPageMargins() specifies default page margins, which are one inch for the top, bottom, left, and right margins, and one-half inch for the gutter, header, and footer margins.

### Output Arguments

#### docxPageMarginsObj — Page margins

DOCXPageMargins object

Page margins, represented by an DOCXPageMargins object.

### Properties

#### Bottom — Bottom margin size

character vector

character vector specifying the width of the bottom margin. The character vector must have the format valueUnits where Units is an abbreviation for the units in which the width size is expressed. Valid abbreviations are:

- no abbreviation — pixels
- cm — centimeters
- in — inches
- mm — millimeters
- pc — picas
- pt — points
- px — pixels

#### Footer — Footer size

character vector

Specify the size using the same format used for the `Bottom` property.

**Gutter — Gutter size**

character vector

Specify the size using the same format used for the `Bottom` property.

**Header — Header size**

character vector

Specify the size using the same format used for the `Bottom` property.

**Id — ID for this document element**

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

**Left — Left margin size**

character vector

Specify the size using the same format used for the `Bottom` property.

**Right — Right margin size**

character vector

Specify the size using the same format used for the `Bottom` property.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

**Top — Top margin size**

character vector

Specify the size using the same format used for the `Bottom` property.

## Examples

**Reset Default Margins**

Reset the margins specified by the default DOM template.

```
import mlreportgen.dom.*;
d = Document('myreport', 'docx');
open(d);

s = d.CurrentDOCXSection;
```

```
s.PageMargins.Left = '.5in';  
s.PageMargins.Right = '.5in';  
append(d, 'Left and right margins are .5 inch');  
  
close(d);  
rptview('myreport', 'docx');
```

### **See Also**

mlreportgen.dom.DOCXSection

### **Topics**

“Report Formatting Approaches” on page 13-17

# mlreportgen.dom.DOCXSection class

**Package:** mlreportgen.dom

Page format and layout for Microsoft Word document section

## Compatibility

---

**Note** mlreportgen.dom.DOCXSection will be removed in a future release. Use mlreportgen.dom.DOCXPageLayout for Word output and mlreportgen.dom.PDFPageLayout for PDF output instead.

---

## Description

Use an mlreportgen.dom.DOCXSection object to define the page format, headers, and footers of a Word document section.

If this is the first DOCXSection in a document, then it controls the page layout of all the document elements from the beginning of a document to this DOCXSection.

If this is the second or later DOCXSection in a document, then it controls the page layout of all the document elements from the preceding DOCXSection to itself.

Before you set properties (such as margin widths) of a DOCXSection object, open the Document object that contains the DOCXSection object.

## Construction

docxSectionObj = DOCXSection() creates an empty document section.

## Output Arguments

### docxSectionObj — Numbering stream counter reset

mlreportgen.dom.DOCXSection object

Page format and layout for Word document section, represented by an mlreportgen.dom.DOCXSection object.

## Properties

### Id — ID for this document element

character vector | string scalar

ID for this document element, specified as a character vector or string scalar. The DOM generates a session-unique ID when it creates the document element. You can specify your own ID.

### PageFooters — Page footers for this layout

array of mlreportgen.dom.DOCXPageFooter objects

You can define up to three page footers for a layout, one each for:

- The first page of the section
- Even pages
- Odd pages

**PageHeaders — Page headers for this layout**

array of `microsoftreportgen.dom.DOCXPageHeader` objects

You can define up to three page headers for a layout, one each for:

- The first page of the section
- Even pages
- Odd pages

**PageMargins — Margin sizes and page orientation in this section**

`microsoftreportgen.dom.DOCXPageMargins` object

Margin sizes and page orientation in this section, specified as an `microsoftreportgen.dom.DOCXPageMargins` object.

**PageSize — Size of pages in this section**

`microsoftreportgen.dom.DOCXPageSize` object

Size of pages in this section, specified as an `microsoftreportgen.dom.DOCXPageSize` object.

**PageBorder — Page borders for this section**

`[]` (default) | `microsoftreportgen.dom.PageBorder` object

Page borders for this section, specified as an `microsoftreportgen.dom.PageBorder` object.

**Parent — Parent of document element**

DOM object

Parent of this document element, specified as a DOM object. This property is read-only.

**RawFormats — XML markup for unsupported section formats**

cell array

Cell array of character vectors, with each character vector containing Word XML markup for a Word format. For information about XML markup for Word formats, see <https://www.ecma-international.org/publications-and-standards/standards/ecma-376/>.

**Style — Formats defining section style**

array of format objects

The formats you specify using this property override corresponding formats defined by the style sheet style specified by the `StyleName` property. The DOM interface ignores formats that do not apply to this element.

**Tag — Tag for this document element**

character vector | string scalar

Tag for this document element, specified as a character vector or string scalar.

The DOM generates a session-unique tag as part of the creation of this object. The generated tag has the form `CLASS:ID`, where `CLASS` is the object class and `ID` is the value of the `Id` property of the



object. Specifying your own tag value can help you to identify where an issue occurred during document generation.

## Examples

### Change Page Margins of a Document Section

Create a Word report. The value of `d.CurrentDOCXSection` is `[]`.

```
import mlreportgen.dom.*;  
d = Document('mydoc', 'docx');
```

Open the document, which generates a `DOCXSection` object from the default template and assigns the handle of the object to `d.CurrentDOCXSection`.

```
open(d);
```

Assign a handle for the document `DOCXSection` object to the `DOCXSection` object `s`.

```
s = d.CurrentDOCXSection;
```

Change the left margin of `s`.

```
s.PageMargins.Left = '0.5in';
```

Add some content and display the report.

```
p = Paragraph('Hello World');  
append(d,p);
```

```
close(d);  
rptview('mydoc.docx');
```

### See Also

`mlreportgen.dom.DocumentPart` | `mlreportgen.dom.DOCXSubDoc` |  
`mlreportgen.dom.DOCXPageFooter` | `mlreportgen.dom.DOCXPageHeader` |  
`mlreportgen.dom.DOCXPageSize` | `mlreportgen.dom.DOCXPageMargins` |  
`mlreportgen.dom.DOCXRawFormat`

### Topics

“Report Formatting Approaches” on page 13-17



# Form-Based Reports

---

- “Form-Based Reports” on page 16-2
- “Create a Simple Form-Based Setup” on page 16-6
- “Report Form” on page 16-13

## Form-Based Reports

### In this section...

“Workflow for Creating Form-Based Reports” on page 16-2

“Create Multiform-Based Report Setups” on page 16-3

“Define Page Layouts in a Form-Based Report Setup” on page 16-4

You can use the Report Explorer to create a report based on a form or a set of forms. A form is a document used to generate other documents of a predetermined type. The form includes the fixed content of the document type and markers to designate the location of the variable content. For example, a tax form contains blanks to indicate the location of variable content.

The Report Explorer provides a set of components that allow you to generate reports from forms represented by Word, HTML, or PDF templates. You can use Word to create forms for Word reports and an HTML or text editor to create forms for HTML and PDF documents. You can use standard features of Word and HTML documents to designate the location of generated content, called holes, in a form.

### Workflow for Creating Form-Based Reports

This is the workflow you use to create a form-based report. For an example that uses this workflow, see “Create a Simple Form-Based Setup” on page 16-6.

#### Create a Template

Create a template to use with Report Explorer.

- 1 In the Report Explorer, create a copy of one of the Report Explorer default templates or a template based on one of the default templates. See “Copy a Template” on page 7-7.
- 2 To specify the form’s fixed content and holes, edit the templates. See:
  - “Open a Template” on page 7-9
  - “Add Holes in a Microsoft Word Template” on page 13-120
  - “Add Holes in HTML and PDF Templates” on page 13-132

#### Create a Form-Based Setup

- 1 To create a Report Form component, in the Report Explorer, select **File > New Form**.
- 2 Use the Report Form component dialog box to specify the output format and report form template.

Specifying the report template populates the Report Form component with **Template Hole** and layout components representing the holes and page layouts defined in the template.

- 3 Use the Report Explorer to append **Paragraph**, **Image**, **Table**, control flow, and other types of components to the **Template Hole** components. At report generation time, the content generated by these components fills the holes in the fixed content specified by the report form template.
- 4 Save the report setup.

## Generate the Report

Executing the Report Form component copies the fixed content specified by its template to the output report. This process fills the holes in the fixed content with content generated dynamically by the children of the Report Form component hole components.

- 1 Select the Report Form component in the Report Explorer.
- 2 To execute the report form, select **File > Report**.

## Create Multiform-Based Report Setups

You can create multiform-based report setups, for example, a report setup that uses separate form templates for the title page, table-of-contents (TOC) section, and chapter sections.

- 1 Create a main template that contains one or more holes to fill with generated content, including content based on other forms, called subforms.
- 2 Create a template for each of the subforms to include in the report. The subforms can contain holes to fill with content, including subforms. Use the same document type for subform templates as you do for the main template.
- 3 Create the report setup's main Report Form component.
- 4 Assign the main template to the main Report Form component. Assigning the template populates the main Report Form component with **Template Hole** component representing the report's top-level holes.
- 5 Populate the main form's **Template Hole** components with components that generate the content for the hole. You can use **Subform** components to fill a hole with content based on subform templates, for example, a title page based on a title page template.

### Fill a Hole with a Subform

You can fill holes in a form-based report with content based on templates called subforms. For example, you can create a title page by filling a hole in the report's main form with a title page subform. If the subform contains holes, you can fill the holes with subforms to generate a report based on a hierarchy of forms.

You can add a **Subform** component to a **Template Hole** component as its child or to a component that is a descendant of the parent **Template Hole** component. For example, if you want to include a subform conditionally or repetitively to a hole, append a control flow component (such as **If**, **For**, or **While**) to the parent **Template Hole** component. Then append the **Subform** component to the control flow component.

- 1 Add a **Subform** component as a child or descendant of the **Template Hole** component representing the hole in the parent form.
- 2 Assign the template that defines the subform to the **Subform** component.

Assigning the subform template to the **Subform** component populates the **Subform** component with **Template Hole** components that represent the holes defined by the subform template.

- 3 Append components to the subform's hole components that generate the content of the holes defined by the subform template. You can also use subforms, that is, **Subform** components, to fill the holes in a subform.

## Define Page Layouts in a Form-Based Report Setup

To define the page layouts of the report generated by the setup, use page layout components in your form-based report setup. The Report Generator provides two page layout components:

- A PDF Page Layout component for PDF reports
- A DOCX Page Layout component for Word reports

Use the appropriate component for your report. If your report has only one page layout, use a single page layout component to define the layout. If your report has multiple sections with differing layouts, use multiple page layout components.

### Generated Page Layout Components from a Template

You can add page layout components to your setup by first defining the page layouts in the templates associated with the Report Form component and the Subform components in your setup. When you assign a template to a Report Form or Subform component in your setup, using the component dialog box, the Report Generator creates a Template Hole component, called a section hole, for each page layout defined in the template. It appends the hole to the associated Form or Subform component as its first (or only) hole. Each section hole contains a page layout component that specifies the corresponding layout for that section defined in the template.

You can override the layout defined in the template by changing the values in the layout component dialog box. You can also add content to the section by using the Report Explorer to append Paragraph and other content components to the section hole component.

If the template defines page headers and footers for the page layout, the Report Generator generates corresponding page header and footer components and appends them to the page layout component. If the template does not define page headers or footers for a page layout, you can use the Report Explorer to append page header and footer components to the page layout component. You can add content to any header and footer by appending content components, for example, Paragraph components, using the Report Explorer.

### Add Page Layouts Interactively to a Report Setup

You do not have to define page layouts in templates. You can define them in your report setup by using the Report Explorer. In this case, append page layout components with the appropriate settings to hole components in your setup. Use this capability if you to create page layouts dynamically at report generation time. For example, you can adjust page margins to fit images whose size you do not know before you generate the report.

### See Also

DOCX Page Layout | PDF Page Layout | Page Footer | Page Header | Subform | Template Hole | `m1reportgen.dom.Document.createTemplate`

### Related Examples

- “Create a Simple Form-Based Setup” on page 16-6
- “Open a Template” on page 7-9
- “Create Page Layout Sections” on page 13-144
- “Generate a Report” on page 5-2

- “Modify Styles in PDF Templates” on page 13-138
- “Modify Styles in a Microsoft Word Template” on page 13-126
- “Modify Styles in HTML Templates” on page 13-137
- “Add Holes in a Microsoft Word Template” on page 13-120
- “Add Holes in HTML and PDF Templates” on page 13-132
- “Fill the Blanks in a Report Form” on page 13-25
- “Simplify Filling in Forms” on page 13-37

## Create a Simple Form-Based Setup

### In this section...

“Create a Word Template” on page 16-6  
 “Create the Report Setup File” on page 16-10  
 “Generate the Report” on page 16-12

This example creates a forms-based setup that generates a report on a set of magic squares (see the `magic` function).

### Create a Word Template


#### Copy the Report Explorer Default Word Template

Templates that you use in the Report Explorer must be copies of the Report Explorer default template or based on a copy.

- 1 Start Report Explorer:
  - report
- 2 In Report Explorer, select **Tools > Edit Document Conversion Template**.
- 3 From the list of templates, select `Default Word Template`. In the dialog box, click **Copy template**.
- 4 Save the template on the MATLAB path and name it `magic-square.dotx`.
- 5 In the list of templates, the new template appears as `Copy of Default Word Template`. Select it and set these properties:
  - **Template id:** `magic-square`
  - **Display name:** `Magic Square`
  - **Description:** Defines the form for my magic square report
- 6 Open the template in Microsoft Word. With the template selected, in the dialog box, click **Open template**.

#### Prepare to Work with Holes in Word

To work with holes in a Word template, display the Word **Developer** ribbon if it is not displayed. Then, in the **Developer** ribbon, turn on design mode. To help you to create content in the right place, display paragraph marks.

- 1 In your Word template, select **File > Options**.
- 2 In the Word Options dialog box, select **Customize Ribbon**.
- 3 From the **Customize the Ribbon** list, under **Main Tabs**, select the **Developer** check box, and click **OK**.
- 4 On the **Developer** ribbon, toggle **Design Mode** on.
- 5 On the **Home** ribbon, click the **Show/Hide Paragraph Marks** button .



## Create Sections and Set Default Headers and Footers

The default template uses a different first page header and footer for the default page layout. For this example, your template must contain only a default page header and footer. Edit the headers and footers so that the template contains only default page layouts.

- 1 With your cursor in the first page, create a section break. From the **Layout** ribbon, select **Breaks > Next Page**.
- 2 On the new page, double-click the page header. In the **Header & Footer Tools** ribbon, toggle **Link to Previous** off.
- 3 Click the footer. Toggle **Link to Previous** off. Click **Close Header and Footer**.
- 4 With your cursor in the second page, create a section break using **Layout > Breaks > Next Page**.
- 5 In the third page, in the header and the footer, toggle **Link to Previous** off.

## Format Page Numbers

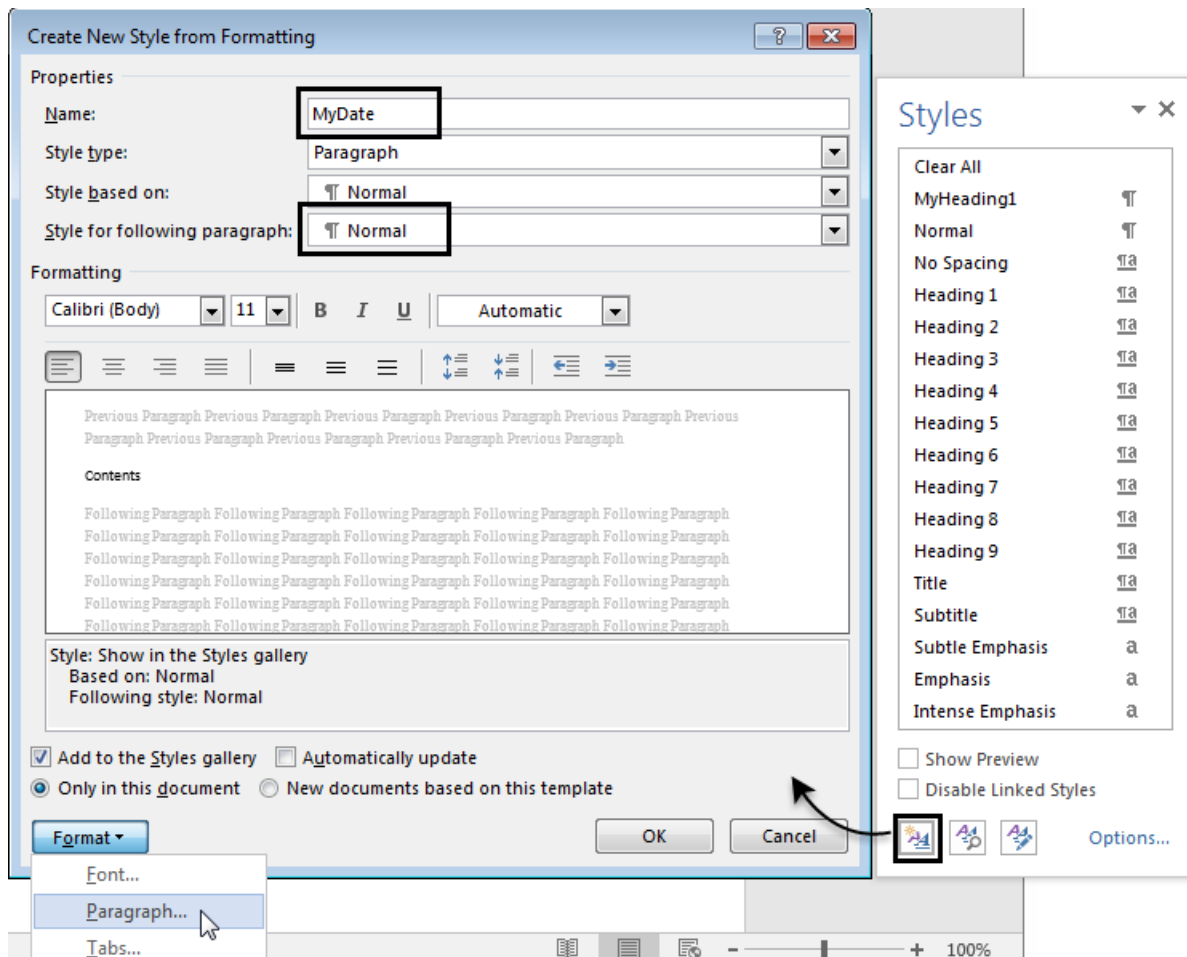
The first section is the title page and does not use a page number. The second section contains the table of contents and uses lowercase roman numerals, starting with page i. The third section is the main content of your report and uses Arabic numerals, starting with page 1.

- 1 In the footer of the first page, delete the page number.
- 2 In the footer of the second page, right-click the page number and select **Format Page Numbers**. Set **Number format** to lowercase roman numerals. Set **Start at** to 1.
- 3 Center the number in the footer.
- 4 In the footer of the last page, right-click the page number and select **Format Page Numbers**. Set **Number format** to Arabic numerals. Set **Start at** to 1.
- 5 Center the number in the footer.
- 6 Click **Close Header and Footer**.

## Add Boilerplate Text and Hole to Title Page

Design a title page that includes boilerplate text and one hole. The boilerplate text is the report title, and the hole is for the date.

- 1 On the first page, before the section break, enter the title text:  
My Magic Square Report
- 2 Apply the Word Title style.
- 3 In a new paragraph, add an inline hole, and give it the title Date.
  - With your cursor in the paragraph, add a space and put the text cursor in front of it. This technique ensures that you are creating an inline hole and not a block hole.
  - On the **Developer** ribbon, in the **Controls** group, click the **Rich Text Content Control** button.
  - On the **Developer** ribbon, click **Properties**. Set **Title** to Date and **Tag** to Hole. Click **OK**.
  - To show the purpose of the hole, enter Date in the hole.
  - Delete the space that follows the hole.
- 4 Create a Word style named MyDate based on Normal.



In the Paragraph properties, change these values:

- **Space before** to 60
- **Alignment** to Centered

5 Apply the MyDate style to the paragraph that contains the Date hole.

### Add the Table of Contents

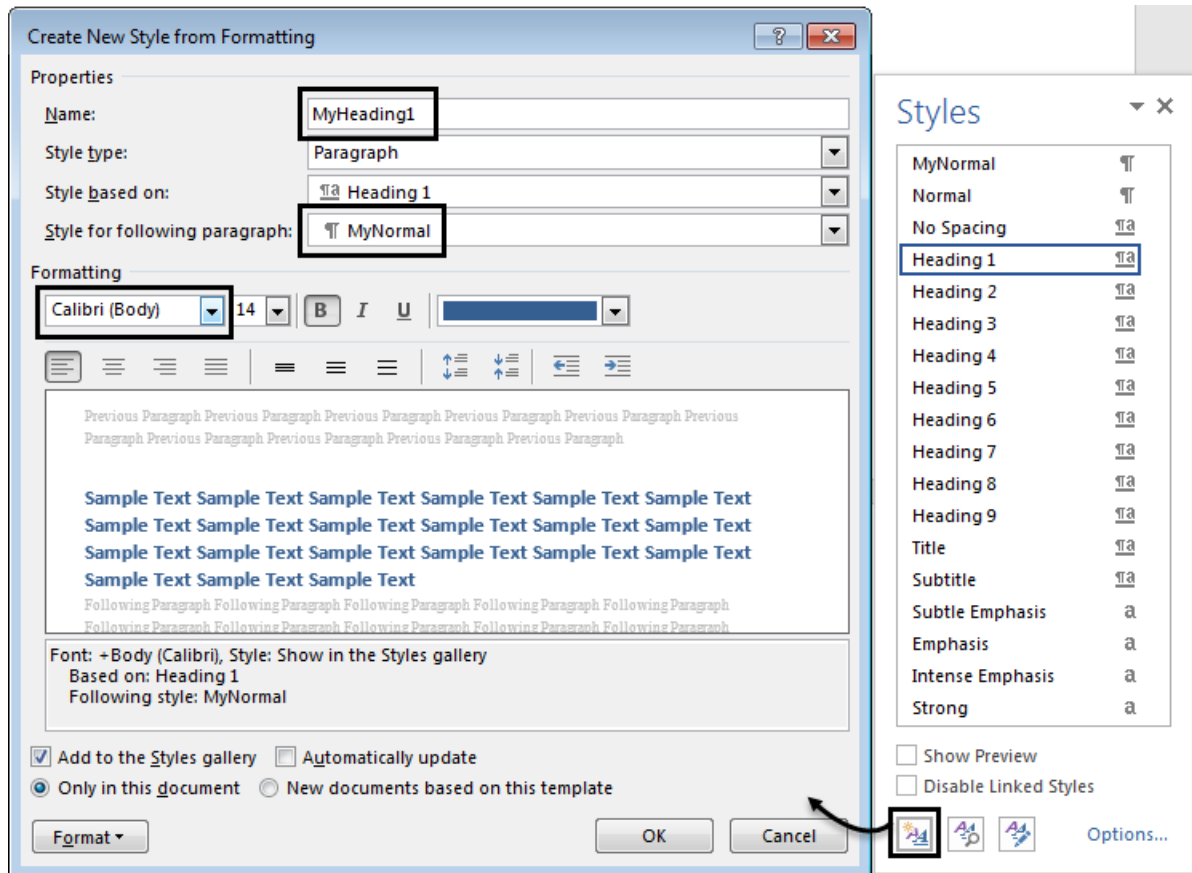
In the second section (the second page), add the table of contents header and field. The TOC contains chapter names and page numbers. The field is a table of contents placeholder that is populated with the chapter names and page numbers when the report is generated.

- 1 Before the section break on the second page, add the heading for the table of contents. Type Contents and apply the style TOC Heading.
- 2 Add a Normal paragraph after the heading and insert the table of contents field. On the **Insert** ribbon, select **Quick Parts > Field**. From the list, select TOC and click **OK**. Respond to the message that appears.

### Add Chapters Hole

The third section of your report is for the main content of the report. Create a block hole and your own heading style to apply to paragraphs in your report setup.

- 1 Insert a block hole and name it Chapters.
  - Select the paragraph marker. Then, on the **Developer** ribbon, click the **Rich Text Content Control** button.
  - With your cursor in the rich text control, on the **Developer** ribbon, click **Properties**. Set **Title** to Chapters and **Tag** to HoLe. Click **OK**.
  - Type Chapters in the hole to indicate the purpose of the hole.
- 2 Create a style MyHeading1 based on Heading 1.



- Change the font to Calibri.
- In the Paragraph formatting, on the **Indents and Spacing** tab, change **Spacing After** to 20.
- On the **Line and Page Breaks** tab, select the **Page Break Before** check box.

### Save and Close the Template

Clean up the template for report generation, and then save and close it.

- 1 Remove any empty paragraphs.
- 2 Turn off paragraph marks.
- 3 Save and close the template.

## Create the Report Setup File

- 1 Create a form-based setup file. In Report Explorer, select **File > New Form**.
- 2 In the Report Form Options, under **Report Output Type and Templates**, change **File format** to Word and change the template name to your custom template, Magic Square.

When you select your custom template, the holes and layouts from your template populate the setup. This setup contains these holes:

- A block hole for the start of the document (first section), named **#start#**. This hole contains a DOCX Page Layout component. Default page header and footer components appear as children of the layout component.
- An inline hole named Date.
- A block hole for the start of the second section, named **#sect2#**, and a block hole for the start of the third section, named **#sect3#**.

Each section hole contains a page layout component. Each page layout component contains a Page Header and Page Footer component to pick up the page number information you specified in the template.

- A block hole named Chapters.
- 3 Add a Text component as a child of the Date inline hole. To insert the current date in the hole, enter this expression in the text box of the Text component. This expression returns the value of the MATLAB date command:

```
%<date>
```

- 4 Create an Eval component as a child of the Chapters Template Hole component in your report. In the component's dialog box, clear the **Insert MATLAB expression in report** and **Display command window output in report** check boxes. Copy this code into the **Expression to evaluate in the base workspace** box:

```
% This M-code sets up two variables
% that define how the report runs.
```

```
% mgicSizeVector is a list of MxM
% Magic Square sizes to insert into
% the report. Magic squares cannot
% be 2x2.
```

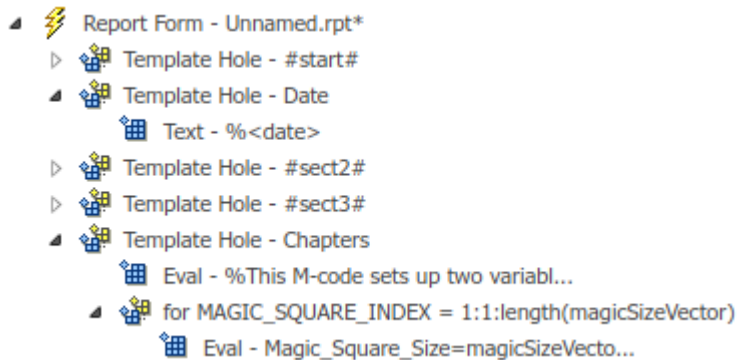
```
magicSizeVector=[4 8 16 32];
```

```
% largestDisplayedArray sets the
% limit of array size to insert
% in the report with the
% Insert Variable component.
```

```
largestDisplayedArray=15;
```

- 5 Create a For Loop component after the Eval component in your report setup. Set the loop's **End** value to `length(magicSizeVector)`. Set **Variable name** to `MAGIC_SQUARE_INDEX`.
- 6 Add an Eval component as the first child of the for loop. In the component dialog box, set **Expression to evaluate** to `Magic_Square_Size=magicSizeVector(MAGIC_SQUARE_INDEX);`. Clear the **Insert MATLAB expression in report** and **Display command window output in report** check boxes.

The figure shows your report setup so far:



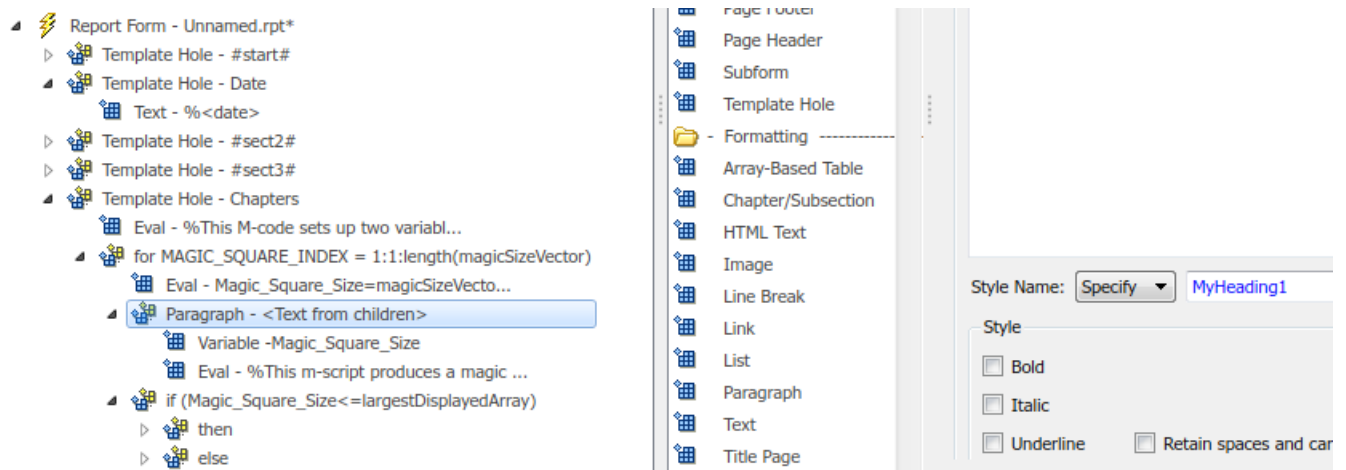
- 7 Create a Paragraph component as a child of the for component. In the dialog box, change the paragraph text **Style Name** to Specify and enter MyHeading1. This setting formats the chapter headings generated by the for loop with the MyHeading1 style you created in your template.
- 8 Create an Insert Variable component as a child of the Paragraph component. Set **Variable name** to Magic\_Square\_Size.
- 9 Create an Eval component after the Variable component. Clear the **Insert MATLAB expression in report** and **Display command window output in report** check boxes. Set **Expression to evaluate** to:

```
% This m-script produces a magic
% square of size Magic_Square_Size
% and creates an image of that square.
```

```
mySquare=magic(Magic_Square_Size);
clf
imagesc(mySquare);
title(sprintf('Magic Square N=%i',Magic_Square_Size))
set(gca,'Ydir','normal');
axis equal;
```

- 10 Create a Logical If component after the Eval component. In the component dialog box, set **Test expression** to Magic\_Square\_Size<=largestDisplayedArray. Move the Logical If component so that it is a sibling of the Paragraph component.
- 11 Create a Logical Then component as a child of the Logical If component. Create an Insert Variable component as a child of the Logical Then component. In the dialog box, set **Variable name** to mySquare.
- 12 Create a Logical Else component following the Logical Then component. Create a Figure Loop component as a child of the Logical Else component.
- 13 Create a Figure Snapshot component as a child of the Figure Loop component. In the Figure Snapshot dialog box, set the paper orientation to portrait. Set **Image size** to Custom: [5 4] inches.

The figure shows the structure of the report setup and the setting for the Paragraph component.



## Generate the Report

Select the Report Form component and select **File > Report**.

## See Also

Template Hole | Paragraph | DOCX Page Layout | Page Footer | Evaluate MATLAB Expression | For Loop | Logical If | Logical Else | Logical Then

## More About

- “Report Form” on page 16-13
- “Form-Based Reports” on page 16-2
- “Add Report Content Using Components” on page 2-11

## Report Form

### In this section...

“Report File Location Options” on page 16-13  
 “Report Output Format” on page 16-14  
 “Report Generation Processing” on page 16-15  
 “Report Description” on page 16-16

This component generates a report based on a form specified by a Word, HTML, or PDF template. To create an instance of this component, in the Report Explorer, select **File > New Form**.

Use this component’s **Report Output Type** and **Templates** options to assign a form template to it. A form template specifies the fixed content of the report generated by this component and holes in the fixed content to fill with generated content. Assigning a form template populates this component with hole and page layout components representing the holes and page layouts of the form template.

You can set the properties of the layout components to override the template layout. You can also append content generation and control flow components to the hole components representing the report form holes.

To execute this component, select **File > Report** in the Report Explorer. Executing this component copies the associated form template’s fixed content to the output report, filling the holes in the fixed content with the content generated by the hole component’s children.

To set the defaults for these options, use the Report Generator preferences. For details, see “Report Generation Preferences” on page 5-9.

## Report File Location Options

### Folder

Use the **Directory** option to specify the folder in which to store the generated report file. Specify a folder to which you have write privileges.

Folder	Option
The same folder as the report setup file	Same as setup file
The current working folder	Present working directory
Temporary folder	Temporary directory
Another folder	Custom. Use the <b>Browse</b> button to select from a list of directories.

You can use %<VariableName> notation to specify a folder in the **Custom** text box.

### Report File Name

Use the **Filename** options to specify a file name for the report file.

File Name	Option
The same file name as the report setup file	Same as setup file (default)
A file name different from the report setup file name	Custom. Enter the name of the report.

You can use %<VariableName> notation to specify a file name in the **Custom** text box.

### Increment to Prevent Overwriting

To maintain the previous version of the setup file when you save updates to the setup file, select **If report already exists, increment to prevent overwriting**.

### Package type

For HTML output, you can specify to generate the report packaging as zipped, unzipped, or both zipped and unzipped.

## Report Output Format

Under **Report Output Type and Templates**, from the **File format** list, select the report output format. For form-based reports, you generate the report in the specified format using a template.

### File Formats

Select one of these options:

- **Direct PDF** — Uses a PDF template.
- **HTML** — Uses an HTML template and produces zipped or unzipped report packaging.
- **PDF (from Word)** — Creates a Word document from a Word template and then generates a PDF document from the Word document.
- **Single-File HTML** — Uses an HTML template and produces a single file.
- **Word** — Uses a Word template.

### Template

To apply a template to this Report Form component, select a template from the list next to the **File format** list. The template list shows all templates on the MATLAB path the first time you opened the Report Explorer. If you added a template to the MATLAB path after opening the Report Explorer for the first time, you must update the list. To update the list, execute this command:

```
rptgen.db2dom.TemplateCache.getTheCache(true);
```

When you select a template, the structure from the template populates the report setup.

- The sections and holes (placeholders you can fill with content) defined in your template appear in the report setup.
- If your template includes a subform template library, these subform templates become available to insert using the **Subform** component. For more information, see “Create Multiform-Based Report Setups” on page 16-3.



For an example that uses templates in form-based reports, see “Create a Simple Form-Based Setup” on page 16-6. For information on customizing templates, see “Create a Report Template” on page 7-6.

## Report Generation Processing

Option	Purpose
<b>View report after generation</b>	<p>When report generation finishes, the viewer associated with the report output format displays the report.</p> <p>On a Linux or Macintosh platform, the report generator calls the <code>soffice</code> command to open the document. On Linux, Apache OpenOffice or LibreOffice must be installed. On Macintosh, Apache OpenOffice must be installed in the <code>/Applications</code> folder.</p> <p>To view the report manually, open the file from the location specified in the <b>Report Options</b> for the report, under <b>Report File Location</b>.</p>
<b>Auto save before generation</b>	Save the report setup file before you generate a report.
<b>Compile model to report on compiled information</b>	<p>Ensure that a report reflects compiled values.</p> <p>By default, the Simulink Report Generator reports uncompiled values of Simulink parameters. The uncompiled values of some parameters, such as signal data types, can differ from the compiled values used during simulation.</p> <p>This option causes the report generator to compile a model before reporting on model parameters. After generating the report, the report generator returns the model to its uncompiled state.</p> <p><b>Note</b> When you select this option, whenever report generation requires simulating the model (for example, the report includes a Model Simulation (Simulink Report Generator) component), the report generator uncompiles the model and then recompiles the model, if necessary, to report on model contents. If a report requires multiple compilations, the processing can be time-consuming.</p> <p>To minimize compilations, consider using separate reports to report on the contents of a model and on the results of simulating that model.</p>
<b>Generate DocBook only</b>	Generate a DocBook XML file and do not generate the Word, PDF, or HTML report.
<b>Evaluate this string after generation</b>	Specify MATLAB code for processing to occur after the report is generated. For example, you can specify to close a model.

## **Report Description**

To record notes and comments about your report setup, use the **Report Description** box.

## **See Also**

PDF Page Layout | DOCX Page Layout | Page Header | Page Footer | Subform | Template Hole

## **More About**

- “Form-Based Reports” on page 16-2
- “Create a Simple Form-Based Setup” on page 16-6

# MATLAB Report Generator Task Examples

---

- “Specify Space Between Paragraphs” on page 17-3
- “Side-by-Side Tables” on page 17-6
- “Fit Wide Tables in a Page” on page 17-8
- “Span a Table Entry Across Rows and Columns” on page 17-13
- “Side-by-Side Images” on page 17-20
- “Side-by-Side Figures” on page 17-22
- “Scale Image to Fit Page” on page 17-24
- “Hyperlink Image” on page 17-26
- “Create a Report with Landscape Pages” on page 17-30
- “Create a Report with Portrait and Landscape Pages” on page 17-38
- “Set Table Column Width” on page 17-43
- “Number Section Headings, Table Titles, and Figure Captions Programmatically” on page 17-46
- “Align Table Entry Content Horizontally” on page 17-51
- “Create a Zebra-Striped Table” on page 17-54
- “Set Page Margins in a Word Report” on page 17-59
- “Set Page Margins in a PDF Report” on page 17-66
- “Programmatically Number Pages” on page 17-73
- “Create an Inline Equation in a Report” on page 17-80
- “Custom Styled Word List” on page 17-82
- “Multilevel List” on page 17-86
- “Number Pages in a PDF Template” on page 17-92
- “Number Pages in a Word Template” on page 17-95
- “Excel to PDF” on page 17-99
- “Prevent MATLAB Figure Display During Report Generation” on page 17-106
- “Create a Table from a Cell Array” on page 17-109
- “Format Numbers in a Table” on page 17-112
- “Center Figure Snapshot on a Page” on page 17-114
- “Center Image on a Page” on page 17-121
- “Center Formal Image in a PDF Report” on page 17-126
- “Define Styles Programmatically” on page 17-131
- “Customize the Page Size and Margins of a Word Report Using Templates” on page 17-135
- “Customize the Page Size and Margins of a PDF Report Using Templates” on page 17-145
- “Customize the Page Size and Margins of a Report Programmatically” on page 17-154

- “Generate a Microsoft Word Document with Page Borders” on page 17-157
- “Generate a PDF Document with Page Borders” on page 17-159
- “Generate a Report API Report with Page Borders” on page 17-161

## Specify Space Between Paragraphs

These examples show how to specify the space between DOM Paragraph objects in a document. To set the spacing, modify a Paragraph object Style property, or create a paragraph style in a document template.

### Setting the Style Programmatically

In this example, paragraph spacing is set programmatically by including OuterMargin objects in the Style property of each paragraph. The following output is created:

#### Spacing Using OuterMargin

This is a paragraph with a bottom outer margin of 50pt.

This is a paragraph with a bottom outer margin of 25pt.

This is a paragraph with a bottom outer margin of 5pt.

Text to show spacing

Import the DOM package so you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
```

Create and open a document. To create a Word document, change the output type from "pdf" to "docx". To create an HTML document, change "pdf" to "html" or "html-file" for a multi-file or single-file document, respectively. Append a heading to describe the example.

```
d = Document("myDoc1", "pdf");
open(d);
```

```
append(d, Heading1("Spacing Using OuterMargin"));
```

Create a paragraph and set its Style to include an OuterMargin object. Set the left, right, and top outer margins to 0 points, and the bottom margin to 50 points. Append the paragraph to the document.

```
p1 = Paragraph("This is a paragraph with a bottom outer margin of 50pt.");
p1.Style = {OuterMargin("0pt", "0pt", "0pt", "50pt")};
```

```
append(d, p1);
```

Create two more paragraphs with different bottom margins. Append the paragraphs to the document along with a final text object so that the bottom margin of the last paragraph can be seen.

```
p2 = Paragraph("This is a paragraph with a bottom outer margin of 25pt.");
p2.Style = {OuterMargin("0pt", "0pt", "0pt", "25pt")};
```

```
append(d, p2);
```

```
p3 = Paragraph("This is a paragraph with a bottom outer margin of 5pt.");
```

```
p3.Style = {OuterMargin("0pt", "0pt", "0pt", "5pt")};  
append(d, p3);
```

```
append(d, Text("Text to show spacing"));
```

Close and view the document.

```
close(d);  
rptview(d);
```

### Using a Style in a Template

In this example, paragraphs use styles defined in a custom template file used by the document. The following output is created:

#### Spacing Using Template-defined Paragraph Styles

This is a paragraph with a custom style defined in a template.

This is a paragraph with the default style defined in a template.

This is a paragraph with a bottom outer margin of 5pt.

Text to show spacing.

Import the DOM package so you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
```

Create and open a PDF document. Specify a custom document template named `exampleTemplate` when creating the Document object. To make Word or HTML documents, first create a Word or HTML template. Then, change "pdf" to "docx", "html", or "html-file" depending on the template created.

See [Create an HTML or PDF Template](#) or [Create a Microsoft Word Template](#) for details on how to create templates.

```
d = Document("myDoc2", "pdf", "exampleTemplate");
```

The custom PDF template `exampleTemplate` has been modified to contain a paragraph style named `exampleParagraphStyle` defined in `exampleTemplate\stylesheets\root.css` as the following:

```
/* Custom paragraph style */  
p.exampleParagraphStyle  
{  
    font-size: 11pt;  
    margin-bottom: 50px;  
}
```

The `margin-bottom` value gives paragraphs a bottom margin of 50 points, similar to the previous example. Alternatively, you can modify the default paragraph style in the template so that paragraphs

in the document automatically have the custom style. The default paragraph style in `exampleTemplate` has been modified to have a bottom margin of 25 points:

```
/* Paragraph */
p {
    font-size: 11pt;
    margin-bottom: 25pt;
}
```

Open the document and append a heading to describe the example.

```
open(d);
```

```
append(d, Heading1("Spacing Using Template-defined Paragraph Styles"));
```

Create a paragraph using `exampleParagraphStyle` as the style name. Create another paragraph with no style name specified so that it uses the default paragraph style in the template.

```
p1 = Paragraph("This is a paragraph with a custom style defined in a template.", ...
    "exampleParagraphStyle");
```

```
p2 = Paragraph("This is a paragraph with the default style defined in a template.");
```

Create another paragraph with the default paragraph style, but set its `Style` property to have a different bottom margin. This will override the template's default style.

```
p3 = Paragraph("This is a paragraph with a bottom outer margin of 5pt.");
p3.Style = {OuterMargin("0pt", "0pt", "0pt", "5pt")};
```

Append the paragraphs to the document along with a final text object so that the bottom margin of the last paragraph can be seen.

```
append(d, p1);
append(d, p2);
append(d, p3);
append(d, Text("Text to show spacing."));
```

Close and view the document.

```
close(d);
rptview(d);
```

## Side-by-Side Tables

This example shows how to arrange tables side-by-side on a page.

The example places each table in adjacent entries of an invisible table, which is a table with no borders or colors. The invisible table causes the inserted tables to appear to be side-by-side.

64	2	3	61	60	6	7	57
9	55	54	12	13	51	50	16
17	47	46	20	21	43	42	24
40	26	27	37	36	30	31	33
32	34	35	29	28	38	39	25
41	23	22	44	45	19	18	48
49	15	14	52	53	11	10	56
8	58	59	5	4	62	63	1

16	2	3	13
5	11	10	8
9	7	6	12
4	14	15	1

Import the DOM and Report API packages so you do not have to use long, fully-qualified class names. Also, create a Report object.

```
import mlreportgen.dom.*
import mlreportgen.report.*

% To create a Word report, change the output type from "pdf" to "docx".
% To create an HTML report, change "pdf" to "html" or "html-file" for a
% multifile or single-file report, respectively.
rpt = Report('myreport', 'pdf');
```

Create two Table objects, table1 and table2, to hold magic squares data. Set the width of these tables to be 100% to fit in the invisible table cells created below. Also, specify the styles for the table borders, row separators, and column separators.

```
tableStyle = { ...
    Width('100%'), ...
    Border('solid','black','1px'), ...
    ColSep('solid','black','1px'), ...
    RowSep('solid','black','1px') ...
};

table1 = Table(magic(8));
table1.TableEntriesHAlign = 'center';
table1.Style = tableStyle;

table2 = Table(magic(4));
```



```
table2.TableEntriesHAlign = 'center';  
table2.Style = tableStyle;
```

Insert the tables in the only row of a 1x3, invisible layout table (lo\_table). A table is considered invisible when the borders are not defined for the table nor any of its table entries.

```
lo_table = Table({table1, ' ', table2});
```

Specify the width of the layout table entries to fit the magic squares tables.

```
lo_table.entry(1,1).Style = {Width('3.2in')};  
lo_table.entry(1,2).Style = {Width('.2in')};  
lo_table.entry(1,3).Style = {Width('3.2in')};
```

Set the layout table width so it spans the width of the page between the margins. Set `ResizeToFitContents` to false so the layout table columns are not resized and instead the specified widths are used.

```
lo_table.Style = {Width('100%'), ResizeToFitContents(false)};
```

Generate and display the report.

```
add(rpt, lo_table);  
close(rpt);  
rptview(rpt);
```

## **Fit Wide Tables in a Page**

This example shows how to fit a wide table in a report.

## Chapter 1. Traffic Cameras in Austin

Repeated Column Index: 1 ,SlicedColumns: From column 2 to column 6

Camera ID	Location Name	Status	Turn on Date	Manufacturer	ATD Location ID
1	830 BLK W RUNDBERG LN (Little Walnut Creek Library, HEB)	TURNED_ON	11/18/2016	Sarix	LOC16-005090
2	AIRPORT BLVD / OAK SPRINGS DR	TURNED_ON	11/8/2016	Spectra	LOC16-000805
3	STAKED PLAINS DR / LAKELINE BLVD	TURNED_ON	12/20/2016	Sarix	LOC16-004915
4	35TH ST / PECOS ST	TURNED_ON	11/30/2016	Spectra	LOC16-003630
5	MARTIN LUTHER KING JR BLVD / RIO GRANDE ST	TURNED_ON	11/18/2016	Sarix	LOC16-000095
6	LAVACA ST / 3RD ST	TURNED_ON	8/10/2016	Sarix	LOC16-001625
7	PEARCE LN / KELLAM RD	TURNED_ON	10/15/2016	Sarix	LOC16-000500
8	WEST GATE BLVD / WILLIAM CANNON DR	TURNED_ON	11/16/2016	Spectra	LOC16-002050
9	ESCARPMENT BLVD / WILLIAM CANNON DR	TURNED_ON	11/16/2016	Spectra	LOC16-002065
10	35TH ST / JACKSON AVE	TURNED_ON	11/16/2016	Spectra	LOC16-000530

Repeated Column Index: 1 ,SlicedColumns: From column 7 to column 11

Camera ID	Landmark	Signal Engineer Area	Council District	Jurisdiction	Location Type
1	Little Walnut Creek Library, HEB	NORTHEAST	4	AUSTIN FULL PURPOSE	BUILDING
2	-nan	NORTHEAST	3,1	AUSTIN FULL PURPOSE	ROADWAY
3	-nan	NORTHWEST	6	AUSTIN FULL PURPOSE	ROADWAY
4	-nan	CENTRAL	10	AUSTIN FULL PURPOSE	ROADWAY
5	-nan	CENTRAL	9	AUSTIN FULL PURPOSE	ROADWAY
6	-nan	CENTRAL	9	AUSTIN FULL	ROADWAY

The data for this example is a mat file containing cell array of traffic camera data from Austin, Texas. This cell array contains information such as camera location, its status and the date when it was turned on etc.

We assume traffic\_data.mat file that contains cell array of traffic camera data is in the current working directory. The requirement is to print the table so all of its columns fit on paper that is 8.5 inches wide by 11 inches long, in portrait orientation.

### Create a Table:

To include a table in a report, use mlreportgen.dom.FormalTable object. This object has a table body and an optional table header and footer.

First, load a mat file containing MATLAB cell array data to workspace. Create a DOM Formal Table object using the cell array data. To make the table easier to read, set the table headings to bold, and add a left margin space between the table column separator and the table content.

```
load('traffic_data.mat');
tbl_header = traffic_camera_data(1,:);
traffic_camera_data(1,:) = [];

formalTable = mlreportgen.dom.FormalTable(tbl_header,traffic_camera_data);
formalTable.RowSep = "Solid";
formalTable.ColSep = "Solid";
formalTable.Border = "Solid";
formalTable.Header.TableEntriesStyle = [formalTable.Header.TableEntriesStyle,...
    {mlreportgen.dom.Bold(true)}];
formalTable.TableEntriesStyle = [formalTable.TableEntriesStyle,...
    {mlreportgen.dom.InnerMargin("2pt","2pt","2pt","2pt"),...
    mlreportgen.dom.WhiteSpace("preserve")}];
```

**Trial Number 1:** Add the DOM Formal Table in a default portrait page of size 8.5 inches wide and 11 inches long.

Import the DOM and Report API packages so you do not have to use long class names.

```
import mlreportgen.dom.*
import mlreportgen.report.*;
```

Create a container to hold the report content.

```
% To create a Word report, change the output type from "pdf" to "docx".
rpt = Report("TrafficCameraDataPortrait","pdf");
```

Create a chapter with the title "Traffic Cameras in Austin".

```
chapter = Chapter("Title","Traffic Cameras in Austin");
```

Add the created table to the chapter and add the chapter to the report.

```
add(chapter, formalTable);
add(rpt, chapter);
close(rpt);
```

**Result:** The generated report includes the table but the table columns are too narrow. Fitting the whole table in a portrait page created a table that is not legible.

**Trial Number 2:** Fit the wide table in a landscape oriented page, which is 11 inches wide by 8.5 inches long.

```
import mlreportgen.dom.*
import mlreportgen.report.*;
import mlreportgen.utils.*

rpt = Report("TrafficCameraDataLandscape","pdf");
chapter = Chapter("Title","Traffic Cameras in Austin");
```

Set the report landscape layout to true. Add the table to the chapter.

```
rpt.Layout.Landscape = true;
add(chapter,formalTable);
add(rpt,chapter);
close(rpt);
```

**Result:** Although the landscape layout is better than the portrait page report, many columns are not legible and the table is not easy to read.

**Trial Number 3:** Use the Report Generator TableSlicer utility to slice the input table into multiple slices. Its MaxCols property specifies the maximum number of columns per table slice.

First, try dividing the table into two slices and print them on default 8.5 wide by 11 inch long portrait paper.

```
import mlreportgen.dom.*
import mlreportgen.report.*;
import mlreportgen.utils.*

rpt = Report("TrafficCameraDataSlicing-1","pdf");
chapter = Chapter("Title","Traffic Cameras in Austin");
```

Now, create a table slicer object and specify the formal table as an input . The input table has 18 columns, so to create two slices, set the MaxCols property to 9.

The table slicer utility has a slice method which slices the input table and generates mlreportgen.utils.TableSlice objects. These objects have the sliced table and the start and end column indices of the original input table.

```
slicer = mlreportgen.utils.TableSlicer("Table",formalTable,"MaxCols",9);
slices = slicer.slice();
```

Use the start and end index to create a customized title. Then add the customized sliced table title and the table slice to the chapter.

```
for slice = slices
    str = sprintf("From column %d to column %d",slice.StartCol,slice.EndCol);
    para = Paragraph(str);
    para.Bold = true;
    para.Style = [para.Style,{KeepWithNext(true),...
        OuterMargin("0pt","0pt","5pt","0pt")}];
    add(chapter,para);
    add(chapter,slice.Table);
end
```

Generate and display the report.

```
add(rpt,chapter);
close(rpt);
```

**Result:** The output is better than first two trials, but the table slices are difficult to read and are disconnected from each other.

**Trial Number 4:** Based on the trial output so far, reduce the MaxCols value to create 4 table slices. Use the RepeatCols property to repeat columns in all the slices. To connect all 4 slices, set the RepeatCols property value to 1 so that the Camera ID column is repeated in every table slice.

```
import mlreportgen.dom.*
import mlreportgen.report.*;
import mlreportgen.utils.*

rpt = Report("TrafficCameraDataSlicing-2","pdf");
chapter = Chapter("Title","Traffic Cameras in Austin");
```

Set the MaxCols value to 6 and the RepeatCols value to 1 .

```
slicer = mlreportgen.utils.TableSlicer("Table",formalTable,"MaxCols",...
    6,"RepeatCols",1);
slices = slicer.slice();
```

Create a customized title using the start and end index. Add the customized sliced table title and the table slice to the chapter.

```
for slice = slices
    str = sprintf("Repeated Column Index: %d ,SlicedColumns: From column %d to column %d",...
        slicer.RepeatCols,slice.StartCol, slice.EndCol);
    para = Paragraph(str);
    para.Bold = true;
    para.Style = [para.Style,{KeepWithNext(true),...
        OuterMargin("0pt","0pt","5pt","0pt")}];
    add(chapter,para);
    add(chapter,slice.Table);
end
```

Generate and display the report.

```
add(rpt,chapter);
close(rpt);
rptview(rpt);
```

**Result:** Output is legible and it satisfies the original requirement to print the table on a portrait page. The input table style, which has bold headers and inner margins that are retained in all the table slices.

The table tile is customized for the readers to understand the table entries data.

*Copyright 2018 The MathWorks, Inc*

## Span a Table Entry Across Rows and Columns

These examples show how to make a table entry span rows and columns.

### Informal Table

This example shows how to use row and column spanning in a DOM informal table object to create the following table.

Name		Address
First	Last	
John	Smith	Natick, MA
Jane	Doe	Boston, MA

An informal table is a table that does not include pre-defined head, body, and foot sections. However, you can format the first few rows of an informal table as a header as this example shows.

Import the DOM package so you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
```

Set up the document and add a heading to describe the table.

```
% To create a PDF report, change the output type from "docx" to "pdf".
% To create an HTML report, change "docx" to "html" or "html-file" for
% a multifile or single-file report, respectively.
rpt = Document('myReport', 'docx');
```

```
h = Heading(1, 'Multiple Row and Column Table Entries Using an Informal Table');
h.Style = [h.Style {HAlign('center')}];
append(rpt, h);
```

Create cell arrays for the styles to be used by the different document components. The containing table spans the width of the page and has solid lines separating the entries. The main header is steel blue, and then sub-header is a lighter sky blue.

```
tableStyle = {Width('100%'), Border('solid'), ColSep('solid'), RowSep('solid')};
mainHeaderRowStyle = {VAlign('middle'), InnerMargin('2pt', '2pt', '2pt', '2pt'), ...
    BackgroundColor('steelblue')};
mainHeaderTextStyle = {Bold, OuterMargin('0pt', '0pt', '0pt', '0pt'), FontFamily('Arial')};
subHeaderRowStyle = {VAlign('middle'), InnerMargin('2pt', '2pt', '2pt', '2pt'), BackgroundColor(
subHeaderTextStyle = {Bold, OuterMargin('0pt', '0pt', '0pt', '0pt'), FontFamily('Arial')};
bodyStyle = {OuterMargin('0pt', '0pt', '0pt', '0pt'), InnerMargin('2pt', '2pt', '2pt', '0pt')};
```

Create some sample data to include in the table. Then, create the Table object.

```
data = {'John', 'Smith', 'Natick, MA';
    'Jane', 'Doe', 'Boston, MA'};
```

```
t = Table(3);
t.Style = [t.Style tableStyle];
```

Create a TableRow object for the first row of the table that contains the Name and Address headers. The Name header has two sub-headers, so it spans two columns. This is set with the ColSpan

property of the table entry. The Address header does not have any sub-headers, so it spans two rows. This is set with the RowSpan property of the table entry.

```
r = TableRow;
r.Style = [r.Style mainHeaderRowStyle];
p = Paragraph('Name');
p.Style = [p.Style mainHeaderTextStyl {HAlign('center')}}];
te = TableEntry(p);
te.ColSpan = 2;
append(r, te);

p = Paragraph('Address');
p.Style = [p.Style mainHeaderTextStyl];
te = TableEntry(p);
te.RowSpan = 2;
append(r, te);
append(t, r);
```

Create a second row in the table for the First and Last sub-headers of Name. Even though the table is three columns wide, the second row contains only two entries because the Address field spans both the first and second rows.

```
r = TableRow;
r.Style = [r.Style subHeaderRowStyle];
p = Paragraph('First');
p.Style = [p.Style subHeaderTextStyl];
te = TableEntry(p);
append(r, te);

p = Paragraph('Last');
p.Style = [p.Style subHeaderTextStyl];
te = TableEntry(p);
append(r, te);
append(t, r);
```

Loop through the sample data to add it to the table.

```
for k = 1:size(data, 1)
    r = TableRow;
    r.Style = [r.Style bodyStyle];
    te = TableEntry(data{k,1});
    append(r, te);
    te = TableEntry(data{k,2});
    append(r, te);
    te = TableEntry(data{k,3});
    append(r, te);
    append(t, r);
end
```

Add the table to the document.

```
append(rpt, t);
```

### Formal Table

This example shows how to use row and column spanning in a DOM formal table object to create the following table.



Name		Address
First	Last	
John	Smith	Natick, MA
Jane	Doe	Boston, MA

A formal table is a table that consists of three sub-tables for the header, body, and footer sections of the table, respectively. Using a formal table instead of an informal table separates the header, which requires special formatting, from the data, which can be passed directly to the formal table constructor to populate the body table.

Import the DOM package so you do not have to use long, fully-qualified class names.

```
import mlreportgen.dom.*
```

Move to a new page in the existing document and add a heading to describe the table.

```
br = PageBreak();
append(rpt, br);
h = Heading(1, 'Multiple Row and Column Table Entries Using a Formal Table');
h.Style = [h.Style {HAlign('center')});
append(rpt, h);
```

Create cell arrays for the styles to be used by the different document components. The containing table spans the width of the page and has solid lines separating the entries. The main header is steel blue, and then sub-header is a lighter sky blue.

```
tableStyle = {Width('100%'), Border('solid'), ColSep('solid'), RowSep('solid')};
mainHeaderRowStyle = {VAlign('middle'), InnerMargin('2pt', '2pt', '2pt', '2pt'), ...
    BackgroundColor('steelblue')};
mainHeaderTextStyle = {Bold, OuterMargin('0pt', '0pt', '0pt', '0pt'), FontFamily('Arial')};
subHeaderRowStyle = {VAlign('middle'), InnerMargin('2pt', '2pt', '2pt', '2pt'), BackgroundColor('skyblue')};
subHeaderTextStyle = {Bold, OuterMargin('0pt', '0pt', '0pt', '0pt'), FontFamily('Arial')};
bodyStyle = {OuterMargin('0pt', '0pt', '0pt', '0pt')};
```

Create some sample data to include in the table. Then, create a FormalTable object with the sample data in the Body section.

```
data = {'John', 'Smith', 'Natick, MA';
    'Jane', 'Doe', 'Boston, MA'};

t = FormalTable(data);
t.Style = [t.Style tableStyle];
t.Body.TableEntriesStyle = [t.Body.TableEntriesStyle, bodyStyle];
```

Construct the Header table of the FormalTable object using the same method as the previous example. Create a TableRow object for the first row of the table that contains the Name and Address headers. The Name header has two sub-headers, so it spans two columns. This is set with the ColSpan property of the table entry. The Address header does not have any sub-headers, so it spans two rows. This is set with the RowSpan property of the table entry.

```
r = TableRow;
r.Style = [r.Style mainHeaderRowStyle];
p = Paragraph('Name');
p.Style = [p.Style mainHeaderTextStyle {HAlign('center')});
```

```
te = TableEntry(p);
te.Style = [te.Style mainHeaderTextStyle];
te.ColSpan = 2;
append(r, te);

p = Paragraph('Address');
p.Style = [p.Style mainHeaderTextStyle];
te = TableEntry(p);
te.Style = [te.Style mainHeaderTextStyle];
te.RowSpan = 2;
append(r, te);
append(t.Header, r);
```

Create a second row in the table for the First and Last sub-headers of Name. Even though the table is three columns wide, the second row only contains two entries because the Address field spans both the first and second rows.

```
r = TableRow;
r.Style = [r.Style subHeaderRowStyle];
p = Paragraph('First');
p.Style = [p.Style subHeaderTextStyle];
te = TableEntry(p);
append(r, te);

p = Paragraph('Last');
p.Style = [p.Style subHeaderTextStyle];
te = TableEntry(p);
append(r, te);
append(t.Header, r);
```

Add the table to the document.

```
append(rpt, t);
```

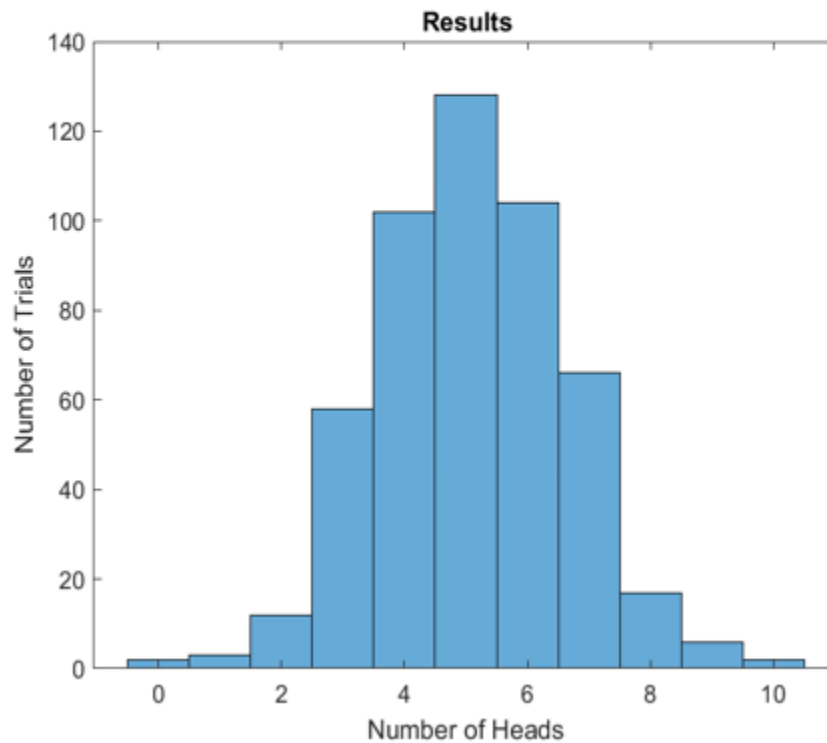
### **Page Layout Table**

This example shows how use row and column spanning to create an invisible page layout table for a complex layout.

## Multi-column Invisible Table

Trial	Heads
1	7
2	6
3	6
4	4
5	6
6	3
7	4
8	8
9	5
10	5
11	6
12	8
13	3
14	3
15	5
16	5
17	0
18	4
19	2
20	5

The plot below shows the results from 500 trials in which a coin was flipped 10 times, and the number of times the coin landed on heads was counted. The table displays the results for the first 20 trials.



Import the DOM package so you do not have to use long, fully-qualified class names.

```
import mlreportgen.dom.*
```

Move to a new page in the existing document and add a heading to describe the table.

```
br = PageBreak();
append(rpt, br);
h = Heading(1, 'Multi-column Invisible Table');
h.Style = [h.Style {HAlign('center')});
append(rpt, h);
```

Determine the styles for the different components on the page. The main table is invisible, so it does not have a border or separators.

```
mainTableStyle = {Width('100%'), Border('none') ColSep('none'), RowSep('none')};
dataTableStyle = {Border('solid'), ColSep('dotted'), RowSep('solid'), ...
    OuterMargin('0pt', '0pt', '0pt', '0pt')};
dataTableEntriesStyle = {OuterMargin('0pt', '0pt', '0pt', '0pt'), VAlign('middle')};
histStyle = {InnerMargin('2pt', '2pt', '2pt', '2pt'), ...
    HAlign('center'), VAlign('bottom'), Width('5in'), Height('4in')};
```

Create example data representing 500 random coin flips. Then, create Text, FormalTable, and Image objects to display information about the data.

```
coinflips = randi(2, [500, 10]);
numHeads = sum(coinflips-1,2);

p = Text(['The plot below shows the results from 500 trials in which a coin was flipped 10 times
        'and the number of times the coin landed on heads was counted. ',...
        'The table displays the results for the first 20 trials.']);

dataHeader = {'Trial', 'Heads'};
dataBody = [(1:20)', numHeads(1:20)];
dataTable = FormalTable(dataHeader, dataBody);
dataTable.TableEntriesStyle = [dataTable.TableEntriesStyle dataTableEntriesStyle];
dataTable.Header.Style = [dataTable.Header.Style {Bold}];
dataTable.Style = [dataTable.Style dataTableStyle];

histogram(numHeads);
title('Results')
xlabel('Number of Heads')
ylabel('Number of Trials')
saveas(gcf, 'histogram_img.png');
close(gcf)
h = Image('histogram_img.png');
```

Create the invisible table and begin adding components. Add the data table to the first row and specify that it spans two rows.

```
t = Table(2);
t.Style = [t.Style mainTableStyle];

row1 = TableRow;
row1.Style = [row1.Style {Width('100%')}];
entry1 = TableEntry;
append(entry1, dataTable);
entry1.RowSpan = 2;
entry1.Style = [entry1.Style {Width("40%")}];
append(row1, entry1);
```

Add the Text object to the first row as well and then append the row to the table.

```
entry2 = TableEntry(p);
entry2.Style = [entry2.Style {Width("60%")}];
append(row1, entry2);

append(t, row1);
```

Create a new row and add the histogram image as the only entry. The data table already fills the first columns of both the first and second rows, so the histogram will be placed in the second column.

```
row2 = TableRow;
entry3 = TableEntry;
h.Style = [h.Style histStyle];
append(entry3, h);
append(row2, entry3);
entry3.Style = [entry3.Style {Width('60%')}];

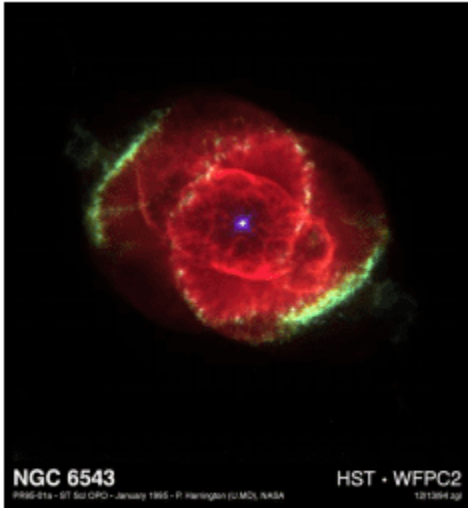
append(t, row2);
```

Generate and display the report.

```
append(rpt,t);  
close(rpt);  
rptview(rpt);
```

## Side-by-Side Images

This example shows how to arrange images side by side on a page.



Import the DOM and Report API packages so you do not have to use long, fully-qualified class names, and create the report.

```
import mlreportgen.dom.*
import mlreportgen.report.*
```

```
% To create a Word report, change the output type from "pdf" to "docx".
% To create an HTML report, change "pdf" to "html" or "html-file" for
% a multifile or single-file report, respectively.
rpt = Report('myreport', 'pdf');
```

Create two image objects wrapped around corresponding image files. Scale the images to fit the invisible table cells created below.

```
imgStyle = {ScaleToFit(true)};
img1 = Image(which('ngc6543a.jpg'));
img1.Style = imgStyle;
img2 = Image(which('peppers.png'));
img2.Style = imgStyle;
```

Insert images in the row of a 1x3, invisible layout table (lot).

```
lot = Table({img1, ' ', img2});
```

The images will be sized to fit the table entries only if their height and width is specified.

```
lot.entry(1,1).Style = {Width('3.2in'), Height('3in')};
lot.entry(1,2).Style = {Width('.2in'), Height('3in')};
lot.entry(1,3).Style = {Width('3.2in'), Height('3in')};
```

Make the table span the width of the page between the margins. Tell the table layout manager to not resize the table columns to fit the images.

```
lot.Style = {ResizeToFitContents(false), Width('100%')};
```

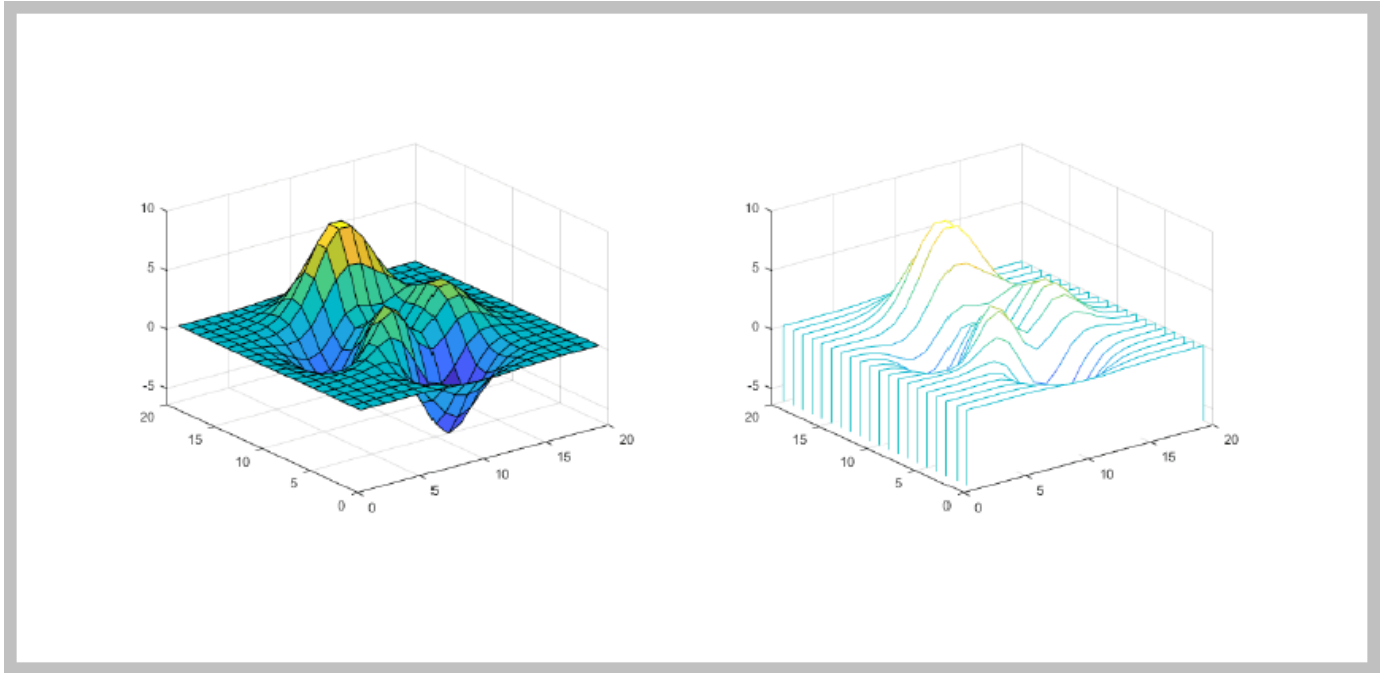
Generate and display the report.

```
add(rpt, lot);  
close(rpt);  
rptview(rpt);
```

## Side-by-Side Figures

This example shows how to arrange figures side-by-side on a page.

The example places each figure in adjacent entries of an invisible table, which is a table with no borders or colors. The invisible table causes the inserted figures to appear to be side-by-side.



Import the DOM and Report API packages so you do not have to use long, fully-qualified class names. Also, create a Report object.

```
import mlreportgen.dom.*
import mlreportgen.report.*
```

```
% To create a Word report, change the output type from "pdf" to "docx".
% To create an HTML report, change "pdf" to "html" or "html-file" for a
% multifile or single-file report, respectively.
```

```
rpt = Report('myreport', 'pdf');
```

Create Figure objects for surface (fig1) and waterfall (fig2) plots. Then, create Image objects wrapped around the figure snapshot image files. Scale the images to fit the table entries created below.

```
imgStyle = {ScaleToFit(true)};
```

```
fig1 = Figure(surf(peaks(20)));
fig1Img = Image(getSnapshotImage(fig1, rpt));
fig1Img.Style = imgStyle;
delete(gcf);
```

```
fig2 = Figure(waterfall(peaks(20)));
fig2Img = Image(getSnapshotImage(fig2, rpt));
fig2Img.Style = imgStyle;
delete(gcf);
```



Insert the images in the only row of a 1x3, invisible layout table (lo\_table). A table is considered invisible when the borders are not defined for the table and all of its table entries.

```
lo_table = Table({fig1Img, ' ', fig2Img});
```

The images will be sized to fit the table entries only if the table entries' height and width are specified.

```
lo_table.entry(1,1).Style = {Width('3.2in'), Height('3in')};  
lo_table.entry(1,2).Style = {Width('.2in'), Height('3in')};  
lo_table.entry(1,3).Style = {Width('3.2in'), Height('3in')};
```

Set the table width so it spans the width of the page between the margins. Set ResizeToFitContents to false so the table columns are not resized and instead the specified widths are used.

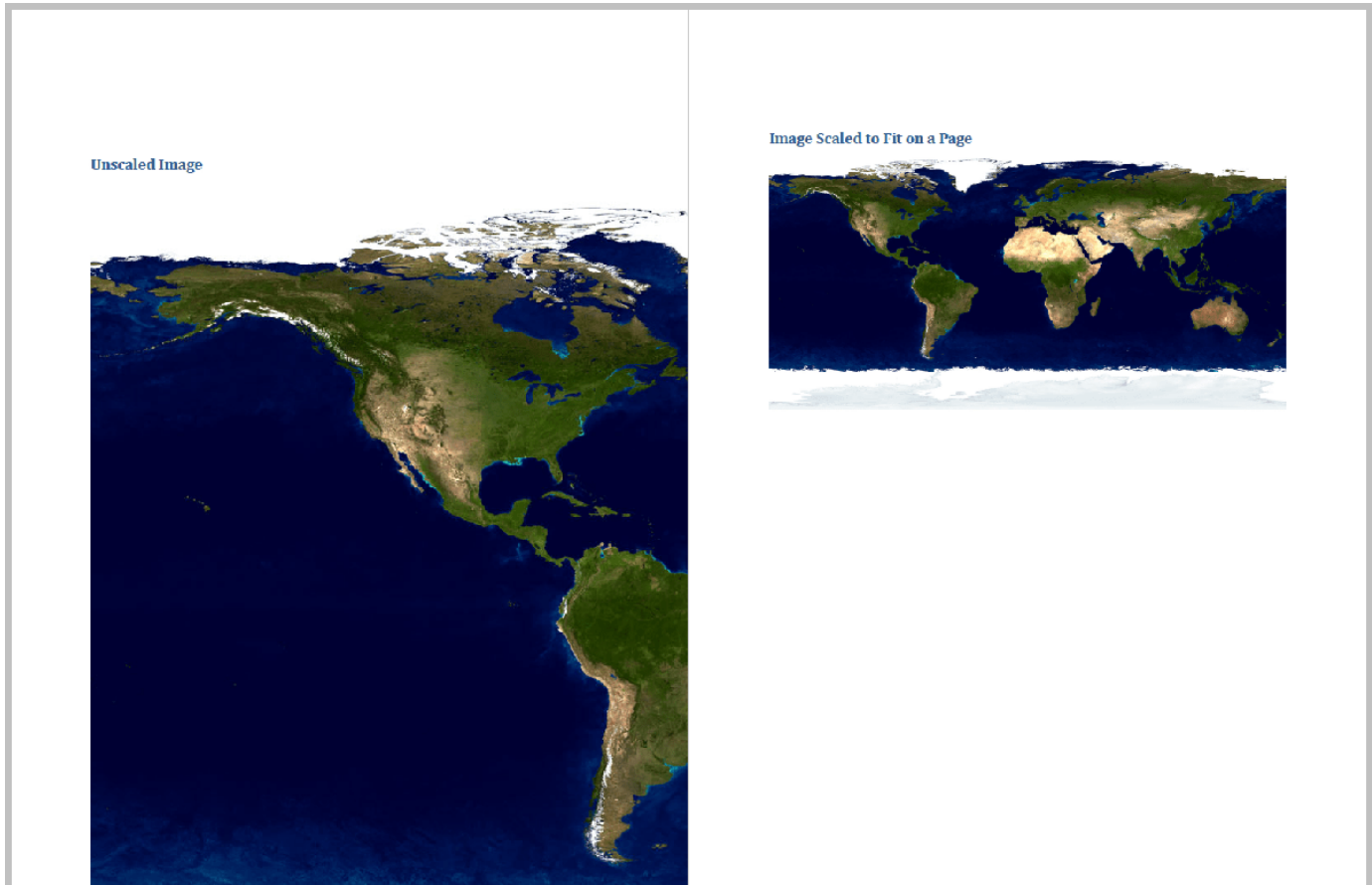
```
lo_table.Style = {Width('100%'), ResizeToFitContents(false)};
```

Generate and display the report.

```
add(rpt, lo_table);  
close(rpt);  
rptview(rpt);
```

## Scale Image to Fit Page

This example shows, for PDF and Word reports, how to scale a large image to fit on a page.



Import the DOM and Report API packages so you do not have to use long, fully-qualified class names.

```
import mlreportgen.dom.*
import mlreportgen.report.*
```

Create and open a report.

```
% To create a Word report, change the output type from "pdf" to "docx".
rpt = Report("myreport", "pdf");
open(rpt);
```

Specify an image that is too large to fit on the page.

```
imgPath = which("landOcean.jpg");
```

Add a heading to the report.

```
heading = Heading1("Unscaled Image");
add(rpt, heading);
```

Add the image to the report using the DOM *Image* class.

```
img1 = Image(imgPath);  
add(rpt,img1);
```

Add a heading to the report.

```
heading = Heading1("Image Scaled to Fit on a Page");  
add(rpt,heading);
```

Use the DOM *ScaleToFit* format to scale the image to fit on the page and then, add the scaled image to the report.

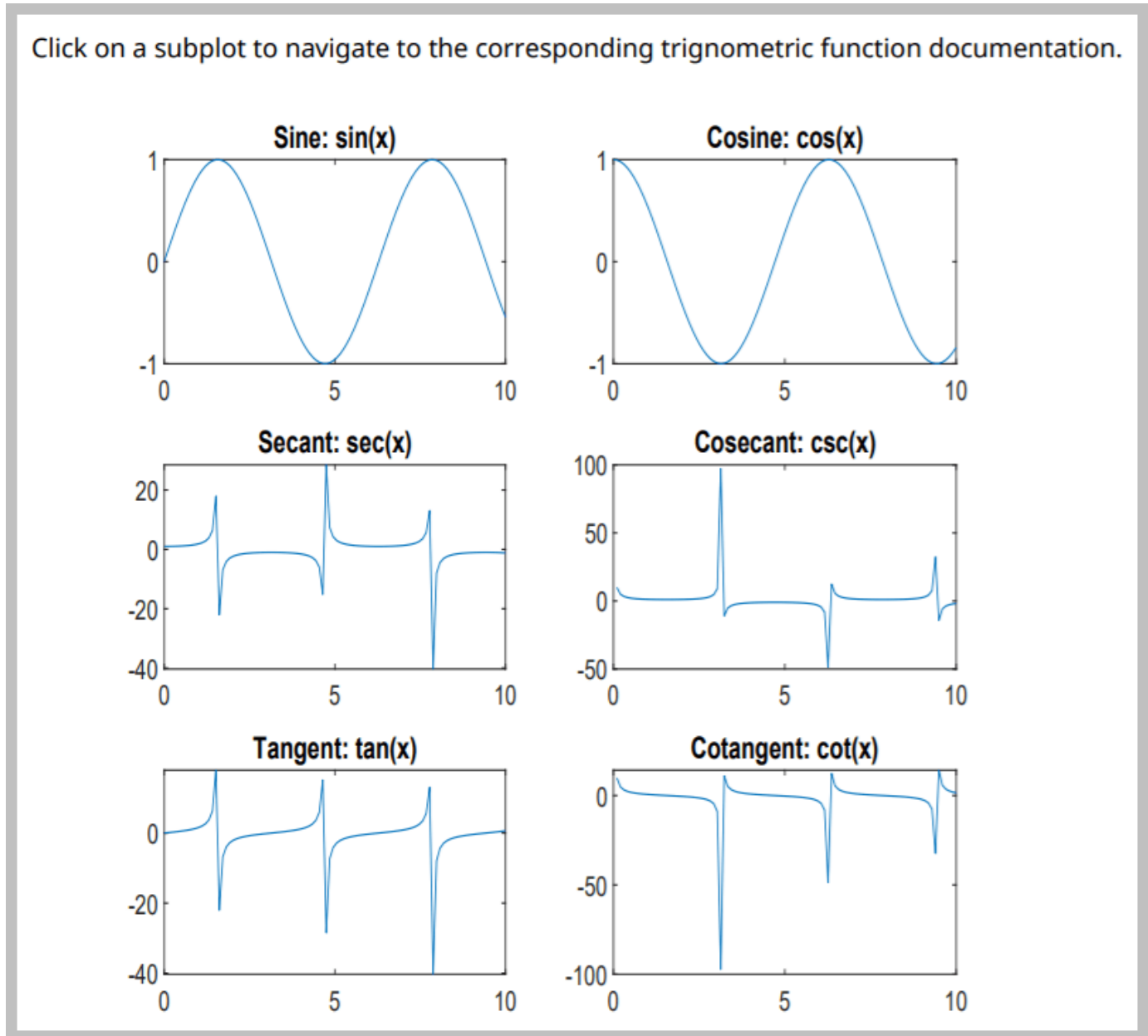
```
img2 = Image(imgPath);  
img2.Style = [img2.Style {ScaleToFit}];  
add(rpt,img2);
```

Close and view the report.

```
close(rpt);  
rptview(rpt);
```

## Hyperlink Image

This example shows how to define areas in an image as hyperlinks. You can define hyperlinks so that they link to a target web page or navigate to another location in the same report.



Import the DOM and Report API packages so you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
import mlreportgen.report.*
```

Create and open a report.

```
% To create an HTML report, change "pdf" to "html" or "html-file" for a
% multifile or single-file report, respectively.
rpt = Report("myreport", "pdf");
open(rpt);
```

Add a paragraph to the report.

```
content = "Click on a subplot to navigate to the corresponding " + ...
         "trigonometric function documentation.";
add(rpt, Paragraph(content));
```

Create a figure with multiple subplots for different trigonometric functions.

```
figH = figure;
x = linspace(0,10);

sinePlot = subplot(3,2,1,"Units","pixels");
plot(x,sin(x));
title("Sine: sin(x)");

cosinePlot = subplot(3,2,2,"Units","pixels");
plot(x,cos(x));
title("Cosine: cos(x)");

secantPlot = subplot(3,2,3,"Units","pixels");
plot(x,sec(x));
title("Secant: sec(x)");

cosecantPlot = subplot(3,2,4,"Units","pixels");
plot(x,csc(x));
title("Cosecant: csc(x)");

tangentPlot = subplot(3,2,5,"Units","pixels");
plot(x,tan(x));
title("Tangent: tan(x)");

cotangentPlot = subplot(3,2,6,"Units","pixels");
plot(x,cot(x));
title("Cotangent: cot(x)");
```

Use the *Figure* reporter to get the figure snapshot. Use the *DOM Image* to include the snapshot in the report.

```
figReporter = Figure("Source",figH,"SnapshotFormat","jpg");
imgPath = getSnapshotImage(figReporter,rpt);

figImg = Image(imgPath);
```

Create an image map on the snapshot with an image area for each subplot. The *getCoords* function, defined at the end of this example, obtains the coordinates of each subplot. The target for a subplot image area is set to be the documentation web page for the trigonometric function that is used to create that subplot. Then, add the snapshot to the report.

```
map = ImageMap;

sinePlotCoords = getCoords(sinePlot);
sinePlotArea = ImageArea("https://www.mathworks.com/help/matlab/ref/sin.html", ...
                        "Sine",sinePlotCoords);
```

```
append(map,sinePlotArea);

cosinePlotCoords = getCoords(cosinePlot);
cosinePlotArea = ImageArea("https://www.mathworks.com/help/matlab/ref/cos.html", ...
    "Cosine",cosinePlotCoords);
append(map,cosinePlotArea);

secantPlotCoords = getCoords(secantPlot);
secantPlotArea = ImageArea("https://www.mathworks.com/help/matlab/ref/sec.html", ...
    "Secant",secantPlotCoords);
append(map,secantPlotArea);

cosecantPlotCoords = getCoords(cosecantPlot);
cosecantPlotArea = ImageArea("https://www.mathworks.com/help/matlab/ref/csc.html", ...
    "Cosecant",cosecantPlotCoords);
append(map,cosecantPlotArea);

tangentPlotCoords = getCoords(tangentPlot);
tangentPlotArea = ImageArea("https://www.mathworks.com/help/matlab/ref/tan.html", ...
    "Tangent",tangentPlotCoords);
append(map,tangentPlotArea);

cotangentPlotCoords = getCoords(cotangentPlot);
cotangentPlotArea = ImageArea("https://www.mathworks.com/help/matlab/ref/cot.html", ...
    "Cotangent",cotangentPlotCoords);
append(map,cotangentPlotArea);

figImg.Map = map;
add(rpt,figImg);
```

Delete the figure window. Close and view the report.

```
close(rpt);
delete(figH);
rptview(rpt);
```

The following function calculates and returns the top-left and bottom-right coordinates for the specified subplot in its parent figure. These coordinates are calculated by converting the subplot axes coordinates, which are based on the figure coordinate system, where its reference point is the bottom-left, to the DOM *ImageArea* coordinate system, where its reference point is the top-left of the image.

```
function coords = getCoords(subplot)
    subplotWidth = subplot.Position(3);
    subplotHeight = subplot.Position(4);

    fig = subplot.Parent;
    figHeight = fig.Position(4);

    x1 = subplot.Position(1);
    y1 = figHeight - (subplot.Position(2) + subplotHeight);

    x2 = x1 + subplotWidth;
    y2 = y1 + subplotHeight;

    coords = [x1, y1, x2, y2];
end
```

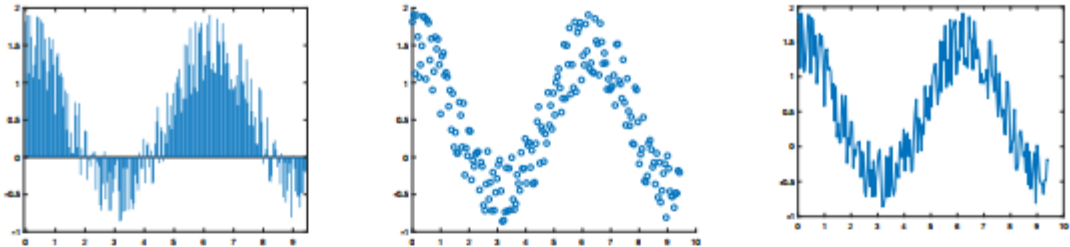


## **Create a Report with Landscape Pages**

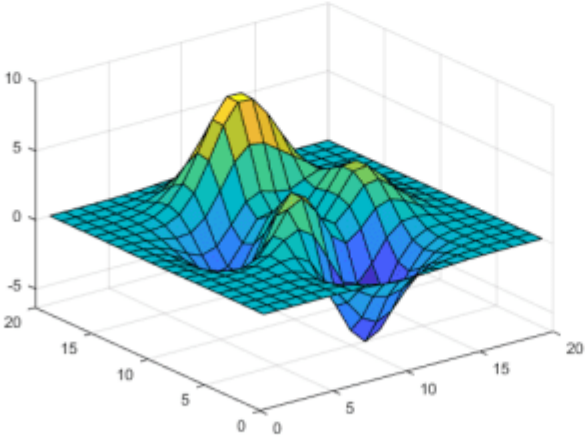
This example shows how to create a report with landscape pages that are 11 inches wide and 8.5 inches high. Using landscape pages allows fitting content that is too wide to fit on a portrait page, such as side-by-side images depicted here.



Chapter 1. Types of Cosine Value Plots with Random Noise



Chapter 2. Surface Plot



Import the DOM and Report API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*;  
import mlreportgen.report.*;
```

Create a container for a PDF report. To create a Word report, change the output type from "pdf" to "docx".

```
rpt = Report("figureSnapshotSideBySideLandscape", "pdf");
```

Set the report landscape layout to true. This sets the entire report layout to landscape.

```
rpt.Layout.Landscape = true;
```

Create a chapter with the title "Types of Cosine Value Plots with Random Noise".

```
chapter = Chapter("Title", "Types of Cosine Value Plots with Random Noise");
```

Create the variables to plot. Create x as 200 equally spaced values between 0 and 3pi. Create y as cosine values with random noise.

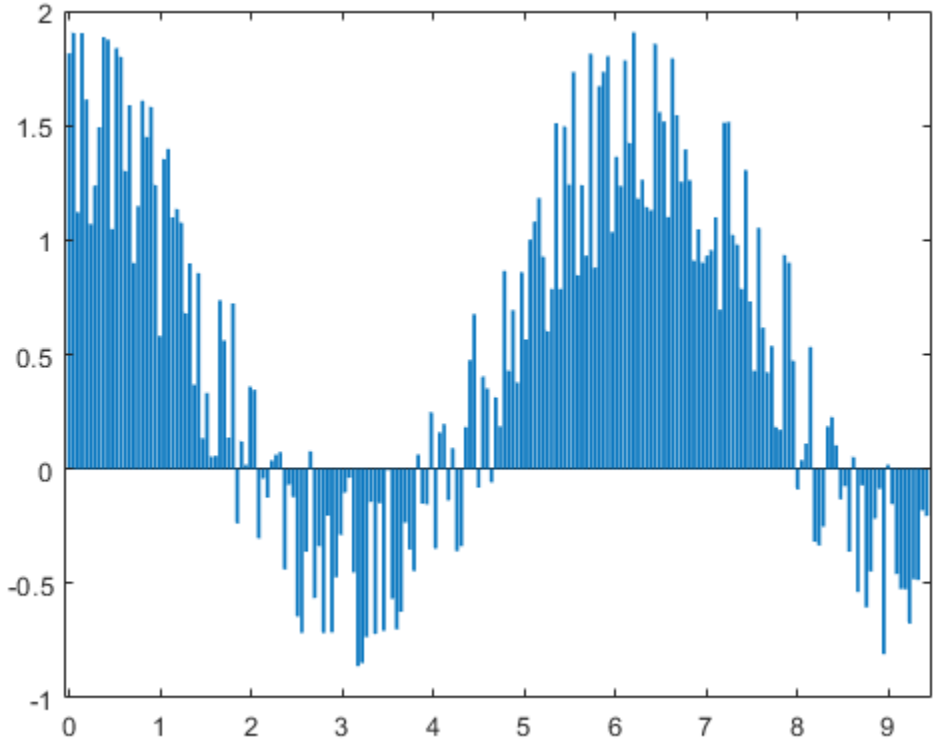
```
x = linspace(0, 3*pi, 200);  
y = cos(x) + rand(1, 200);
```

Create figure objects of the x and y values: bar graph (fig1), scatter plot (fig2) and 2-D Line plot (fig3).

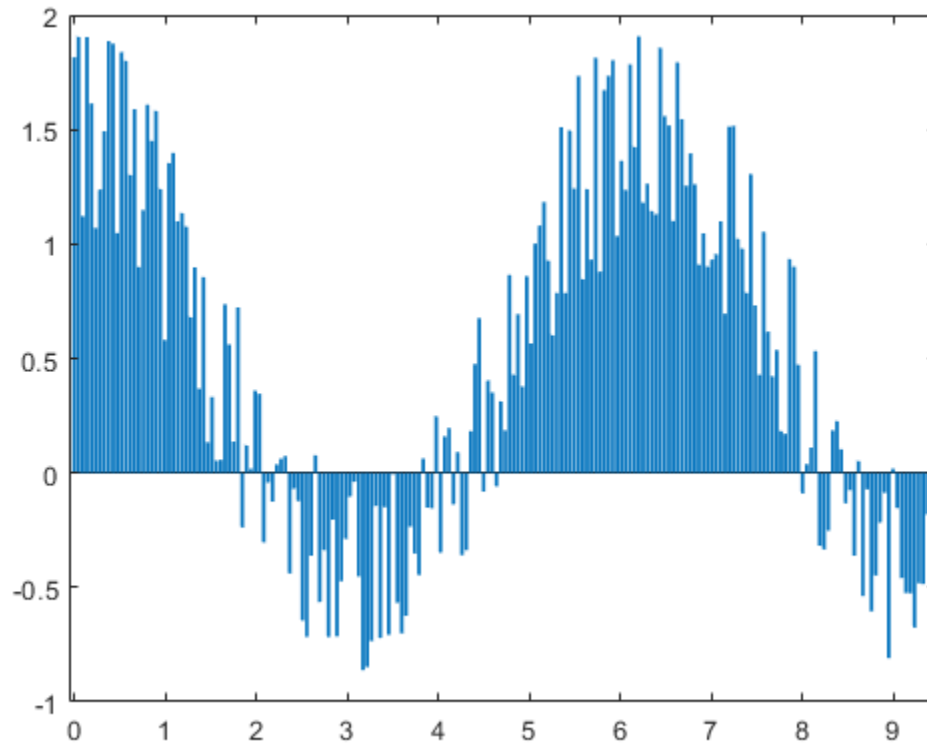
Create image objects wrapped around the figure snapshot image files. Set the scaling of the figure objects so they fit in the table entries.

```
imgStyle = {ScaleToFit(true)};
```

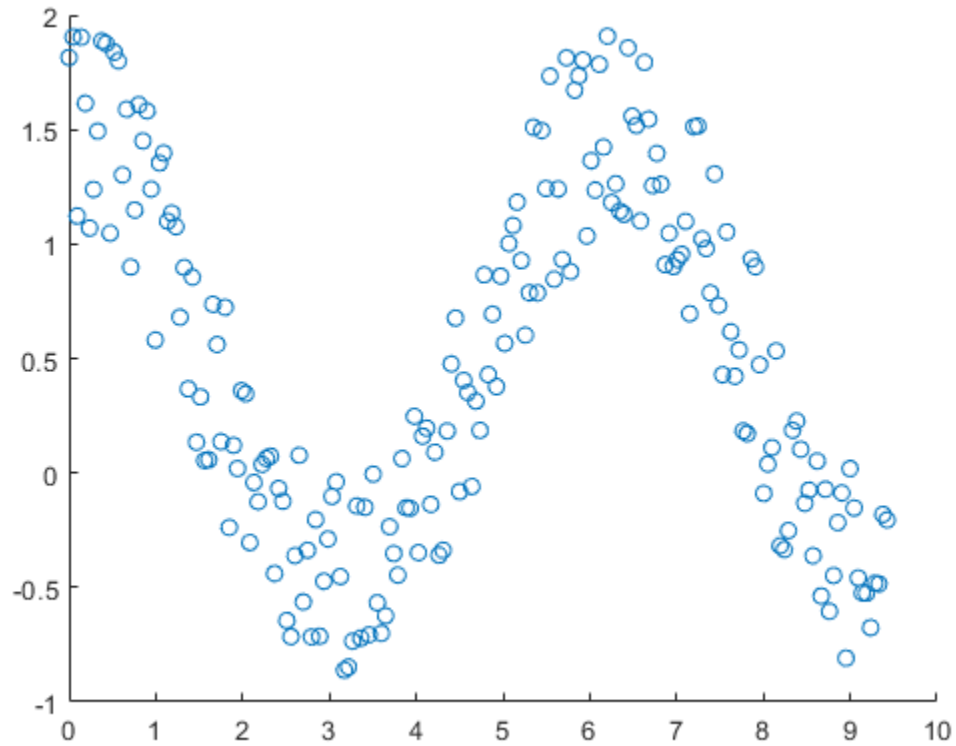
```
fig1 = Figure(bar(x, y));
```



```
fig1Img = Image(getSnapshotImage(fig1, rpt));
```

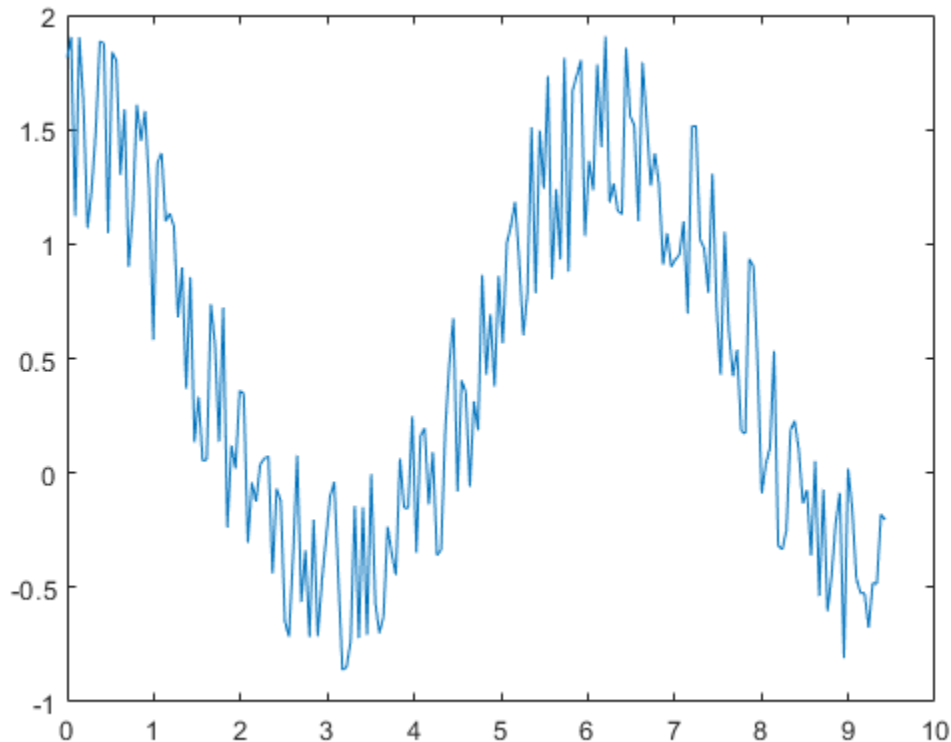


```
fig1Img.Style = imgStyle;  
delete(gcf);  
  
fig2 = Figure(scatter(x,y));
```



```
fig2Img = Image(getSnapshotImage(fig2, rpt));  
fig2Img.Style = imgStyle;  
delete(gcf);
```

```
fig3 = Figure(plot(x,y));
```



```
fig3Img = Image(getSnapshotImage(fig3, rpt));
fig3Img.Style = imgStyle;
delete(gcf);
```

Insert the images in the only row of a 1x5 invisible layout table(`lo_table`)(space between figures by having 2 empty table entries). A table is considered invisible when the borders are not defined for the table nor any of its table entries. The images are sized to fit the table entries only if the height and width of table entries are specified.

```
lo_table = Table({fig1Img, ' ', fig2Img, ' ',fig3Img});
lo_table.entry(1,1).Style = {Width('3.2in'), Height('3in')};
lo_table.entry(1,2).Style = {Width('.2in'), Height('3in')};
lo_table.entry(1,3).Style = {Width('3.2in'), Height('3in')};
lo_table.entry(1,4).Style = {Width('.2in'), Height('3in')};
lo_table.entry(1,5).Style = {Width('3in'), Height('3in')};
```

Add the table to the chapter and the chapter to the report.

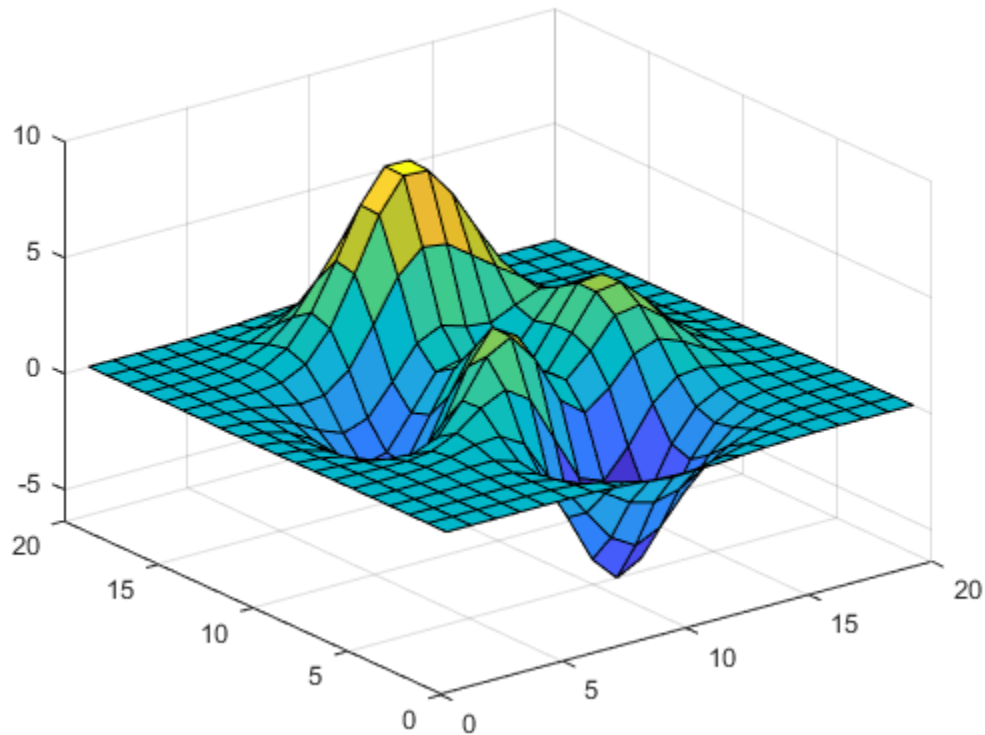
```
add(chapter, lo_table);
add(rpt, chapter);
```

Create a chapter with the title "Surface Plot".

```
chapter1 = Chapter("Title", "Surface Plot");
```

Create a figure object for surface plot (`fig4`). Create image objects wrapped around the figure snapshot image files.

```
fig4 = Figure(surf(peaks(20)));
```



```
fig4Img = Image(getSnapshotImage(fig4, rpt));  
fig4Img.Style = imgStyle;  
delete(gcf);
```

Add the generated image object to the chapter and the chapter to the report.

```
add(chapter1, fig4Img);  
add(rpt, chapter1);
```

Generate and display the report

```
close(rpt);  
rptview(rpt);
```

The generated report includes the side-by-side figure snapshots and the surface plot on landscape pages. The generated side-by-side figure snapshots are clearly legible.

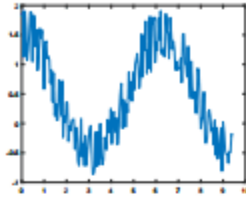
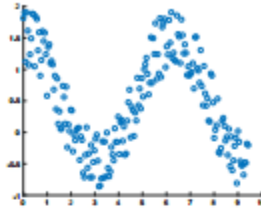
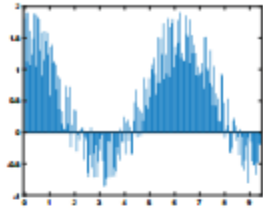
## **Create a Report with Portrait and Landscape Pages**

This example shows how to create a report that contains both 11 x 8.5-in landscape and 8.5 x 11-in portrait pages. The report uses a landscape page to fit content that is too wide to fit on a portrait page, such as these side-by-side images.



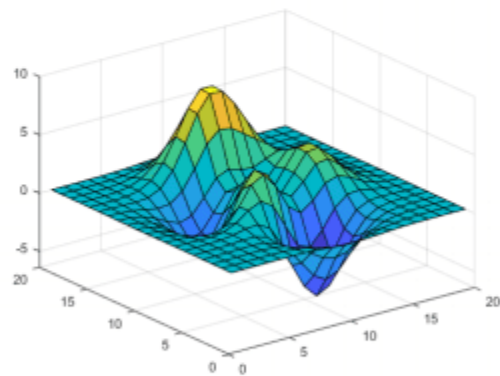
---

Chapter 1. Types of Cosine Value Plots with Random Noise



---

Chapter 2. Surface Plot



Import the DOM and Report API packages so you do not have to use long class names.

```
import mlreportgen.dom.*;
import mlreportgen.report.*;
```

Create a container to hold the report content.

```
% To create a Word report, change the output type from "pdf" to "docx".
rpt = mlreportgen.report.Report("PortraitAndLandscapeReport", "pdf");
```

Create a chapter with the title "Types of Cosine Value Plots with Random Noise".

```
chapter = Chapter("Title", "Types of Cosine Value Plots with Random Noise");
```

Set the chapter landscape layout to true. This will set the entire chapter layout to landscape.

```
chapter.Layout.Landscape = true;
```

Create the variables to plot. Create x as 200 equally spaced values between 0 and 3pi. Create y as cosine values with random noise.

```
x = linspace(0,3*pi,200);
y = cos(x) + rand(1,200);
```

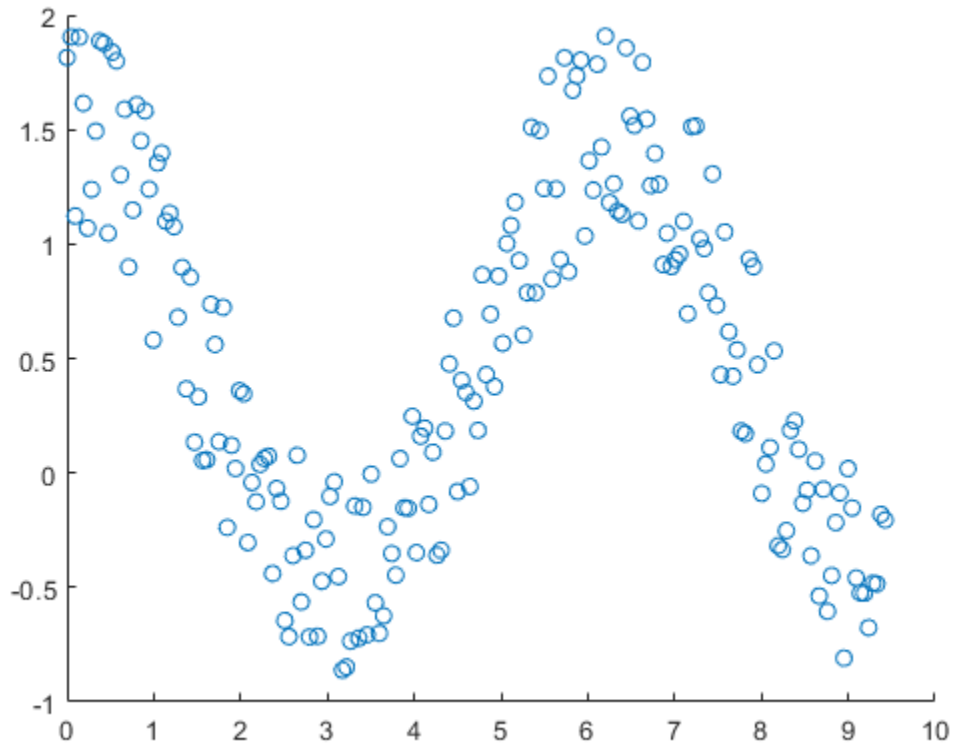
Create figure objects of the x and y values: bar graph (fig1), scatter plot (fig2) and 2-D Line plot (fig3).

Create image objects wrapped around the figure snapshot image files. Set the scaling of the figure objects so they fit in the table entries.

```
imgStyle = {ScaleToFit(true)};

fig1 = Figure(bar(x, y));
fig1Img = Image(getSnapshotImage(fig1, rpt));
fig1Img.Style = imgStyle;
delete(gcf);

fig2 = Figure(scatter(x,y));
```



```
fig2Img = Image(getSnapshotImage(fig2, rpt));
fig2Img.Style = imgStyle;
delete(gcf);
```

```
fig3 = Figure(plot(x,y));
fig3Img = Image(getSnapshotImage(fig3, rpt));
fig3Img.Style = imgStyle;
delete(gcf);
```

Insert the images in the only row of a 1x5 invisible layout table(`lo_table`)(space between figures by having 2 empty table entries). A table is considered invisible when the borders are not defined for the table nor any of its table entries. The images are sized to fit the table entries only if the height and width of table entries are specified.

```
lo_table = Table({fig1Img, ' ', fig2Img, ' ',fig3Img});
lo_table.entry(1,1).Style = {Width('3.2in'), Height('3in')};
lo_table.entry(1,2).Style = {Width('.2in'), Height('3in')};
lo_table.entry(1,3).Style = {Width('3.2in'), Height('3in')};
lo_table.entry(1,4).Style = {Width('.2in'), Height('3in')};
lo_table.entry(1,5).Style = {Width('3in'), Height('3in')};
```

Add the table to the chapter and the chapter to the report.

```
add(chapter, lo_table);
add(rpt, chapter);
```

Create a chapter with the title "Surface Plot". The default layout for a chapter is portrait.

```
chapter1 = Chapter("Title", "Surface Plot");
```

Create a figure object for surface plot (fig4). Create image objects wrapped around the figure snapshot image files.

```
fig4 = Figure(surf(peaks(20)));  
fig4Img = Image(getSnapshotImage(fig4, rpt));  
fig4Img.Style = imgStyle;  
delete(gcf);
```

Add the generated image object to the chapter and the chapter to the report.

```
add(chapter1, fig4Img);  
add(rpt, chapter1);
```

Generate and display the report

```
close(rpt);  
rptview(rpt);
```

The generated report includes the side-by-side figure snapshots on a landscape page and the surface plot on a portrait page.

## Set Table Column Width

This example shows how to set column widths for a DOM Table and FormalTable.

### Set column width for a DOM Table

First Name	Last Name	Address
John	Smith	Natick, MA
Jane	Doe	Boston, MA
Robert	Stewart	Natick, MA

### Set column width for a DOM FormalTable

First Name	Last Name	Address
John	Smith	Natick, MA
Jane	Doe	Boston, MA
Robert	Stewart	Natick, MA

Import the DOM API package so you do not have to use long, fully-qualified class names.

```
import mlreportgen.dom.*
```

Create and open a document. To create a Word document, change the output type from pdf to docx. To create an HTML document, change pdf to html or html - file for a multifile or single-file document, respectively.

```
d = Document("mydoc",  );
open(d);
```

Define table, table entries, and header row styles, which will be used in the later sections.

```
tableStyle = ...
{ ...
Width("100%"), ...
Border("solid"), ...
RowSep("solid"), ...
ColSep("solid") ...
};

tableEntriesStyle = ...
{ ...
HAlign("center"), ...
VAlign("middle") ...
};

headerRowStyle = ...
{ ...
InnerMargin("2pt", "2pt", "2pt", "2pt"), ...
BackgroundColor("gray"), ...
};
```

```
Bold(true) ...
};
```

Define content for the table header row and table body, which will be used later to create a three-column DOM Table and a FormalTable.

```
headerContent = ...
{ ...
'First Name', 'Last Name', 'Address' ...
};

bodyContent = ...
{ ...
'John', 'Smith', 'Natick, MA'; ...
'Jane', 'Doe', 'Boston, MA'; ...
'Robert', 'Stewart', 'Natick, MA' ...
};
```

The following code uses the DOM `TableColSpecGroup` to define styles for a group of columns in a table. Setting the `Span` property to 3 applies the group formatting to all three columns of the table. To format one or more adjacent table columns in the group, use the DOM `TableColSpec` objects. The first `TableColSpec` object, `specs(1)`, spans to the first 2 columns in the group. Set the `Width` format in its `Style` property to make each of these two columns 20% of the table width. The second `TableColSpec` object, `specs(2)`, spans to a single column, that is, the third column in the group, and is formatted to be 60% of the table width.

```
grps(1) = TableColSpecGroup;
grps(1).Span = 3;

specs(1) = TableColSpec;
specs(1).Span = 2;
specs(1).Style = {Width("20%")};

specs(2) = TableColSpec;
specs(2).Span = 1;
specs(2).Style = {Width("60%")};

grps(1).ColSpecs = specs;
```

The following code appends a heading and a DOM Table to the document. Assigning the `grps`, created in the above code, to the `ColSpecGroups` property of the table, makes the first 2 columns 20% and the third column 60% of the table width. The code also assigns styles for the table, table entries, and the first row of the table.

```
append(d,Heading1("Set column width for a DOM Table"));

tableContent = [headerContent; bodyContent];

table = Table(tableContent);
table.ColSpecGroups = grps;

table.Style = tableStyle;
table.TableEntriesStyle = tableEntriesStyle;

firstRow = table.Children(1);
firstRow.Style = headerRowStyle;

append(d,table);
```

The following code appends a heading and a DOM `FormalTable` to the document. Assigning the `grps` to the `ColSpecGroups` property of the formal table makes the first 2 columns 20% and the third column 60% of the table width. The code also assigns styles for the formal table, table entries, and the header row of the formal table.

```
append(d,Heading1("Set column width for a DOM FormalTable"));
```

```
formalTable = FormalTable(headerContent,bodyContent);  
formalTable.ColSpecGroups = grps;
```

```
formalTable.Style = tableStyle;  
formalTable.TableEntriesStyle = tableEntriesStyle;
```

```
headerRow = formalTable.Header.Children;  
headerRow.Style = headerRowStyle;
```

```
append(d,formalTable);
```

Close and view the document.

```
close(d);  
rptview(d);
```

## Number Section Headings, Table Titles, and Figure Captions Programmatically

This example shows how to programmatically create numbered headings for chapters and hierarchical numbered headings for subsections in a chapter. The example also demonstrates how to create hierarchical numbered table titles and figure captions that reside in a numbered chapter or a subsection in a chapter.

<a href="#">Chapter 1. Figures with numbered captions</a> .....	2
<a href="#">1.1. Figure for Land Ocean</a> .....	2
<a href="#">1.2. Figures for Peppers and Cat's Eye Nebula</a> .....	3
<a href="#">Chapter 2. Tables with numbered titles</a> .....	5
<a href="#">2.1. Table for Magic(5)</a> .....	5
<a href="#">2.2. Tables for Magic(8) and Magic(10)</a> .....	5

Import the DOM API package so you do not have to use long, fully-qualified class names.

```
import mlreportgen.dom.*
```

Create and open a document. To create a Word document, change the output type from pdf to docx. To create a HTML document, change pdf to html or html - file for a multifile or single-file document, respectively.

```
d = Document("mydoc", );
open(d);
```

Append a table of contents to the document.

```
append(d, TOC);
```

Create numbering streams for chapter headings, subsection headings, figure captions, and table titles. By default, the streams are created using Arabic numbers with an initial value of 0.

```
chapterStream = createAutoNumberStream(d, "chapter");
sectionStream = createAutoNumberStream(d, "section");
figureStream = createAutoNumberStream(d, "figure");
tableStream = createAutoNumberStream(d, "table");
```

Define some image, table, and table entries styles, which will be used in the later sections.

```
imageStyle = ...
    { ...
      Height("5in"), ...
      Width("5in") ...
    };

tableStyle = ...
    { ...
      Width("100%"), ...
      Border("solid"), ...
      RowSep("solid"), ...
    };
```



```

    ColSep("solid") ...
  };

tableEntriesStyle = ...
  { ...
  HAlign("center"), ...
  VAlign("middle") ...
  };

```

The following code creates the first chapter in the document. The chapter title is created using the `getChapterTitle` function. This function uses the chapter numbering stream to create a numbered chapter title. The chapter consists of two subsections, whose titles are created using the `getSectionTitle` function. This function uses both chapter and section numbering streams to create the hierarchical numbered section title. The subsections consists of multiple figures with hierarchical numbered captions, which are created using the `getFigureCaption` function. This function uses both chapter and figure numbering streams to create the hierarchical numbered figure caption. The `getChapterTitle`, `getSectionTitle`, and `getFigureCaption` functions used to create this chapter are described later in this example.

```

% Chapter 1.
chapterTitle = getChapterTitle("Figures with numbered captions");
append(d,chapterTitle);

% Section 1.1.
sectionTitle = getSectionTitle("Figure for Land Ocean");
append(d,sectionTitle);

% Figure 1.1.
image1 = Image(which("landOcean.jpg"));
image1.Style = imageStyle;
append(d,image1);
append(d,getFigureCaption("Land Ocean"));

% Section 1.2.
sectionTitle = getSectionTitle("Figures for Peppers and Cat's Eye Nebula");
append(d,sectionTitle);

% Figure 1.2.
image2 = Image(which("peppers.png"));
image2.Style = imageStyle;
append(d,image2);
append(d,getFigureCaption("Peppers"));

% Figure 1.3.
image3 = Image(which("ngc6543a.jpg"));
image3.Style = imageStyle;
append(d,image3);
append(d,getFigureCaption("Cat's Eye Nebula or NGC 6543"));

```

The following code creates the second chapter in the document. The chapter has a numbered title with two hierarchical numbered subsections. Here, the subsections consists of multiple tables with hierarchical numbered titles, which are created using the `getTableTitle` function, defined later in this example. This function uses both chapter and table numbering streams to create the hierarchical numbered table title.

```

% Chapter 2.
chapterTitle = getChapterTitle("Tables with numbered titles");

```

```
append(d,chapterTitle);

% Section 2.1.
sectionTitle = getSectionTitle("Table for Magic(5)");
append(d,sectionTitle);

% Table 2.1.
append(d,getTableTitle("Magic(5)"));
table1 = Table(magic(5));
table1.Style = tableStyle;
table1.TableEntriesStyle = tableEntriesStyle;
append(d,table1);

% Section 2.2.
sectionTitle = getSectionTitle("Tables for Magic(8) and Magic(10)");
append(d,sectionTitle);

% Table 2.2.
append(d,getTableTitle("Magic(8)"));
table2 = Table(magic(8));
table2.Style = tableStyle;
table2.TableEntriesStyle = tableEntriesStyle;
append(d,table2);

% Table 2.3.
append(d,getTableTitle("Magic(10)"));
table3 = Table(magic(10));
table3.Style = tableStyle;
table3.TableEntriesStyle = tableEntriesStyle;
append(d,table3);

Close and view the document.

close(d);
rptview(d);
```

The following function returns a numbered title for a chapter. The numbered title is created using the DOM `Heading1` object, where the title content is prefixed by a string `Chapter N` and a period, where `N` is the chapter stream counter. For example, the title for the first chapter contains "Chapter 1." as its prefix. The `CounterInc` format in the `Style` property causes the chapter stream counter to be incremented when this chapter title is appended to the document. The `CounterReset` format in the `Style` property causes the other associated stream counters such as, `section`, `figure`, and `table`, to be reset to their initial values when this chapter title is appended to the document.

```
function chapterTitle = getChapterTitle(content)
    import mlreportgen.dom.*

    chapterTitle = Heading1();
    append(chapterTitle,Text("Chapter "));
    append(chapterTitle,AutoNumber("chapter"));
    append(chapterTitle,Text(". "));
    append(chapterTitle,Text(content));

    chapterTitle.Style = ...
        { ...
          CounterInc("chapter"), ...
          CounterReset("section figure table"), ...
          WhiteSpace("preserve"), ...
```

```

    PageBreakBefore(true), ...
    KeepWithNext(true) ...
  };
end

```

The following function returns a hierarchical numbered title for a section that is in a chapter. The hierarchical numbered title is created using the DOM Heading2 object, where the title content is prefixed by a string "N.M.", where N and M are the chapter and section stream counters, respectively. For example, the title for the first section in the second chapter contains "2.1" as its prefix. The CounterInc format in the Style property causes the section stream counter to be incremented when this section title is appended to the document.

```

function sectionTitle = getSectionTitle(content)
  import mlreportgen.dom.*

  sectionTitle = Heading2();
  append(sectionTitle,AutoNumber("chapter"));
  append(sectionTitle,Text("."));
  append(sectionTitle,AutoNumber("section"));
  append(sectionTitle,Text(". "));
  append(sectionTitle,Text(content));

  sectionTitle.Style = ...
  { ...
    CounterInc("section"), ...
    WhiteSpace("preserve"), ...
    KeepWithNext(true) ...
  };
end

```

The following function returns a hierarchical numbered caption for a figure added to a chapter or a subsection in a chapter. The hierarchical numbered caption is created using the DOM Paragraph object, where the caption content is prefixed by a string "Figure N.F.", where N and F are the chapter and figure stream counters, respectively. For example, the caption for the third figure in the second chapter contains "Figure 2.3." as its prefix. The CounterInc format in the Style property causes the figure stream counter to be incremented when this figure caption is appended to the document.

```

function figureCaption = getFigureCaption(content)
  import mlreportgen.dom.*

  figureCaption = Paragraph();
  append(figureCaption,Text("Figure "));
  append(figureCaption,AutoNumber("chapter"));
  append(figureCaption,Text("."));
  append(figureCaption,AutoNumber("figure"));
  append(figureCaption,Text(". "));
  append(figureCaption,Text(content));

  keepWithPrevious = FOProperty("keep-with-previous.within-page","always");
  figureCaption.Style = ...
  { ...
    CounterInc("figure"), ...
    WhiteSpace("preserve"), ...
    FOProperties(keepWithPrevious) ...
  };
end

```

The following function returns a hierarchical numbered title for a table added to a chapter or a subsection in a chapter. The hierarchical numbered title is created using the DOM Paragraph object, where the title content is prefixed by a string "Table N.T.", where N and T are the chapter and table stream counters, respectively. For example, the title for the third table in the second chapter contains "Table 2.3." as its prefix. The CounterInc format in the Style property causes the table stream counter to be incremented when this table title is appended to the document.

```
function tableTitle = getTableTitle(content)
    import mlreportgen.dom.*

    tableTitle = Paragraph();
    append(tableTitle,Text("Table "));
    append(tableTitle,AutoNumber("chapter"));
    append(tableTitle,Text("."));
    append(tableTitle,AutoNumber("table"));
    append(tableTitle,Text(". "));
    append(tableTitle,Text(content));

    tableTitle.Style = ...
        { ...
        CounterInc("table"), ...
        WhiteSpace("preserve"), ...
        KeepWithNext(true), ...
        Bold(true), ...
        OuterMargin("0pt", "0pt", "10pt", "5pt") ...
        };
end
```

## Align Table Entry Content Horizontally

This example shows how to specify the horizontal alignment of table entries in DOM API informal and formal tables. The example shows how to set the alignment of all table entries, the entries in a row, and an individual entry.

### All Table Entries Centered

Item	Quantity	Cost
Table	1	55
Chair	4	100
Bookshelf	2	40

### Top Row Entries Centered

Item	Quantity	Cost
Table	1	55
Chair	4	100
Bookshelf	2	40

### Header Entries Centered, One Footer Entry Right-Aligned

Item	Quantity	Cost
Table	1	55
Chair	4	100
Bookshelf	2	40
<b>Total</b>		195

### Document and Data Setup

Import the DOM package so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
```

Create and open a PDF document. To create a Word document instead, change the output type from pdf to docx. To create an HTML document instead, change pdf to html or html-file for a multi-file or single-file document, respectively.

```
d = Document("mydoc", "pdf");
open(d);
```

Define the styles for the table, header row, and footer row.

```
tableStyle = { Width("80%"), ...
               Border("solid"), ...
               RowSep("solid"), ...
               ColSep("solid") };

headerStyle = { BackgroundColor("LightBlue"), ...
                Bold(true) };

footerStyle = { BackgroundColor("LightCyan"), ...
```

```
ColSep("none"), ...  
WhiteSpace("preserve" )};
```

Define content for the table header row, body, and footer row.

```
headerContent = {'Item', 'Quantity', 'Cost'};  
bodyContent = {'Table', 1, 55; ...  
              'Chair', 4, 100; ...  
              'Bookshelf', 2, 40};
```

```
total = sum([bodyContent{:},3]);  
footerContent = {[], 'Total ', total};
```

```
tableContent = [headerContent; bodyContent];
```

### Set Alignment of All Table Entries

For the first table, center all of the table entries by setting a single table property.

Create an `mlreportgen.dom.Table` object using the previously defined content. Assign styles to the table and the first row of the table.

```
append(d,Heading1("All Table Entries Centered"));
```

```
table = Table(tableContent);  
table.Style = tableStyle;
```

```
firstRow = table.Children(1);  
firstRow.Style = headerStyle;
```

Set the horizontal alignment of all table entries by setting the `TableEntriesHAlign` property to "center". Append the table to the document.

```
table.TableEntriesHAlign = "center";  
append(d, table);
```

### Set Alignment of Entries in a Table Row

For the next table, center only the top row table entries by modifying the row `Style` property.

Assign styles to the table and the first row of the table. Instead of setting the horizontal alignment with the `TableEntriesHAlign` property, create an `HAlign` object and include it in the `Style` property of the first row along with the header style defined previously.

```
append(d,Heading1("Top Row Entries Centered"));
```

```
table = Table(tableContent);  
table.Style = tableStyle;
```

```
firstRow = table.Children(1);  
firstRow.Style = [{HAlign("center")},headerStyle];
```

```
append(d, table);
```

The alignment defined for a table row by the `Style` property of the table row overrides the alignment defined for the entire table by the `TableEntriesHAlign` property.

## Set Alignment of a Formal Table Header and a Single Entry in the Footer

For the last table, create a formal table using an `mlreportgen.dom.FormalTable` object. Center the table header entries and align a single footer entry to the right.

Create an `mlreportgen.dom.FormalTable` object. Set the styles of the overall table, header, and footer. Because the header of a `FormalTable` object is separate from the body, you can set the alignment of the header entries by using the `TableEntriesHAlign` property without affecting the body or footer entries. Alternatively, you can add an `HAlign` object to the row `Style` property.

```
append(d, Heading1("Header Entries Centered, One Footer Entry Right-Aligned"));
```

```
table = FormalTable(headerContent,bodyContent,footerContent);
table.Style = tableStyle;
```

```
table.Header.TableEntriesHAlign = "center";
table.Header.Style = headerStyle;
```

```
footer = table.Footer;
footer.Style = footerStyle;
```

Align the second entry of the footer to the right by including an `HAlign` object in the `Style` property of the entry. Make the entry bold by adding an `mlreportgen.dom.Bold` object.

```
totalEntry = entry(footer,1,2);
totalEntry.Style = {HAlign("right"),Bold(true)};
```

```
append(d,table);
```

Close and view the document.

```
close(d);
rptview(d);
```

## See Also

`mlreportgen.dom.Table` | `mlreportgen.dom.FormalTable` | `mlreportgen.dom.HAlign`

## More About

- “Choose Type of Table to Create” on page 13-62
- “Create Informal Tables” on page 13-64
- “Create Formal Tables” on page 13-66
- “Format Tables” on page 13-72

## Create a Zebra-Striped Table

This example shows how to create a table with alternating color rows or columns. These tables are called zebra-striped or banded tables. To create a zebra-striped table in a report, you can define it in a program or a template. The examples in this section have zebra-striped rows. Use a similar technique for zebra-striped columns.

The Report Generator APIs support creating zebra-striped tables programmatically or using a Word or HTML template. You cannot create a PDF report for a zebra-striped table using a PDF template.

64	2	3	61	60	6	7	57
9	55	54	12	13	51	50	16
17	47	46	20	21	43	42	24
40	26	27	37	36	30	31	33
32	34	35	29	28	38	39	25
41	23	22	44	45	19	18	48
49	15	14	52	53	11	10	56
8	58	59	5	4	62	63	1

### Zebra-Striped Table Using a Program

This program creates an 8-by-8 magic square table. It has row background colors that alternate between blue and white, which can be helpful for reading and summing the rows. The program also includes formatting for the row height, table width, borders, and alignment of the cell entries.

```
import mlreportgen.report.*
import mlreportgen.dom.*

rpt = Report('zebraTable', 'pdf');

maglen = 8;
mag = magic(maglen);

tb = Table(mag);

% Set the colors for alternating rows
for i = 1:maglen
    r = tb.row(i);
    if mod(i,2)==0
        r.Style = {BackgroundColor('lightsteelblue')};
    else
        r.Style = {BackgroundColor('white')};
    end
end

tb.Style={RowHeight('0.3in'),RowSep('solid'),ColSep('solid')};
```



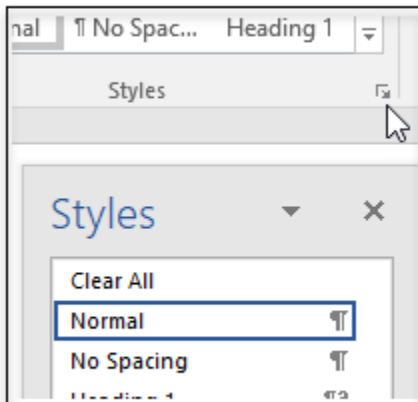
```
tb.Width= '3in';
tb.TableEntriesVAlign = 'middle';
tb.TableEntriesHAlign = 'center';
tb.Border = 'single';
```

```
add(rpt,tb)
close(rpt)
rptview(rpt)
```

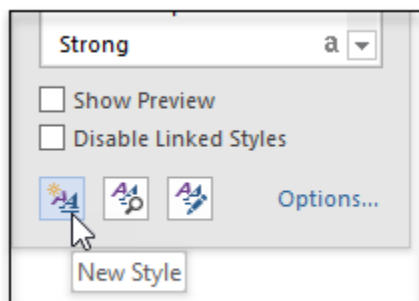
### Zebra-Striped Table Using a Word Template

This example shows how to add a table style to a Word template that defines a zebra-striped table. Using a template modularizes your application. Instead of updating the program, which may introduce bugs, you can update the template.

1. Open a Word template. In this example, the template file is myrpt.dotx, which you can create using `mlreportgen.report.Report.createTemplate('myrpt','docx')`. To open a Word template file, right-click the file and then, click Open in the menu. (If you click the file directly, a .doc file that uses that template opens.)
2. Open the **Styles** pane as shown.



3. In the Styles pane, click the **New Style** button.



4. To define your table style, specify or select the field values. To match the programmatic zebra-striped table example, set these fields to apply the features to the table and table rows:

- **Name** - Add ZebraStripeTable as the name for the style. Use this style name to specify the style to use for the table in your program.
- **Style Type** - Table
- **Apply formatting to** - Even banded rows
- **Color field** (No Color) - Select a color for the odd banded rows from the dropdown.

Then, set these fields to apply these additional features to the whole table:

- **Apply formatting to** - Whole table
- Alignment - Align Center
- Borders - All Borders

**Create New Style from Formatting**

**Properties**

Name: ZebraStripeTable

Style type: Table

Style based on: Table Normal

**Formatting**

Apply formatting to: Even banded rows

Font style: B I U Font color: Automatic

Font size: 1/2 pt Font weight: Automatic

Alignment: Center

Color: Blue

Borders: All Borders

	Jan	Feb	Mar	Total
East	7	7	5	19
West	6	4	7	17
South	8	7	9	24
Total	21	18	21	60

Centered  
Line spacing: single, Space  
After: 0 pt, Center, Box: (Single solid line, Auto, 0.5 pt Line width)  
Pattern: Clear (Accent 1), Priority: 100

Only in this document  New documents based on this template

Format OK Cancel

5. Click **OK** to save the new style.

6. Save the template file

7. In your program, specify the template file to use, and then, you can apply the new zebra-stripe style to a table in your program.

```
rpt = mlreportgen.dom.Document('myreport', 'docx', 'myrpt.dotx');
tb = Table();
tb.StyleName = 'ZebraStripeTable';
```

Not all formatting options that you can use in a program are available in Word. For this example to match the programmatic example, in addition to specifying styles in the Word template, you must specify the row height and table width in your program.

```
tb.Style = {RowHeight('0.3in')};
tb.Width = '3in';
```

This is the complete code for using the Word template, `myrpt.dotx`, to format a magic square as a zebra-striped table.

```
import mlreportgen.report.*
import mlreportgen.dom.*

rpt = mlreportgen.report.Report('myreport', 'docx', 'myrpt.dotx');
maglen = 8;
mag = magic(maglen);

tb = Table(mag);
tb.StyleName = 'ZebraStripeTable';
tb.Style={RowHeight('0.3in')};
tb.Width= '3in';

add(rpt,tb)
close(rpt);
rptview(rpt)
```

### Zebra-Striped Table Using an HTML Template

This example shows how to add a table style to an HTML template that defines a zebra-striped table. Using a template modularizes your application. Instead of updating the program, which may introduce bugs, you can update the template.

1. If you do not have an existing HTML template, create one using `mlreportgen.report.Report.createTemplate('myrpt', 'html')`. In this example, the template file is in a zipped template package, `myrpt.htmix`.
2. Use `unzipTemplate('myrpt.htmix')` to unzip the template to create a folder named `myrpt`, which contains the style sheets and image template files.
3. Go to the `stylesheets` folder in the `myrpt` folder. Open the `root.css` file in a text editor.
4. Create a CSS rule that defines a style name `ZebraStripeTable` for an HTML table element. To define the CSS rule for the `ZebraStripeTable` style, add the following lines to the `root.css` file. The background colors, `#BOC4DE` and `#FFFFFF`, are light blue and white, respectively.

```
/* Settings for whole table */
table.ZebraStripeTable {
    text-align: center;
```

```
        border: 1px solid black;
        border-collapse: collapse;
        width: 5in;
        height: 4in;
    }
    /* Settings for table body */
    table.ZebraStripeTable td {
        padding: 0pt 0pt 0pt 0pt;
        vertical-align: middle;
        text-align: center;
        border: 1px solid black;
        border-collapse: collapse;
    }
    /* Zebra rows and colors */
    tr:nth-child(even) {
        background-color: #B0C4DE
    }
    tr:nth-child(odd) {
        background-color: #FFFFFF
    }
}
```

5. Save the `root.css` file.

6. Use `zipTemplate('myrpt')` to zip the template files back to the `myrpt.htm` template package.

7. In your program, specify `ZebraStripedTable` as the style of your table.

```
rpt = mlreportgen.report.Report('myreport', 'html', 'myrpt.htm');
tb = Table();
tb.StyleName = 'ZebraStripeTable';
```

This is the complete code for using the HTML template, `myrpt.htm`, to format a magic square as a zebra-striped table.

```
import mlreportgen.report.*
import mlreportgen.dom.*

rpt = mlreportgen.report.Report('myreport', 'html', 'myrpt.htm');

maglen = 8;
mag = magic(maglen);
tb = Table(mag);
tb.StyleName = 'ZebraStripeTable';

add(rpt, tb);
close(rpt);
rptview(rpt);
```


## **Set Page Margins in a Word Report**

This example shows how to define page margins in a Word(DOCX) report. You can define top, bottom, left, right margins for a DOCX page, plus its header, footer and gutter sizes.

Sample Traffic Data in Austin

Camera ID	Status	Manufacturer	Signal Engineer Area
1	TURNUED_ON	Spectra	NORTHEAST
2	TURNUED_ON	Sarix	NORTHWEST
3	TURNUED_OFF	Spectura	SOUTHWEST
3	TURNUED_ON	Spectura	NORTHEAST
4	TURNUED_ON	Sarix	SOUTHEAST
5	TURNUED_ON	Spectra	NORTHEAST
6	TURNUED_ON	Sarix	NORTHWEST
7	TURNUED_OFF	Spectura	SOUTHWEST
8	TURNUED_ON	Spectura	NORTHEAST
9	TURNUED_ON	Sarix	SOUTHEAST

---



### Create Report

Import the DOM API packages so you do not have to use long, fully-qualified class names.

```
import mlreportgen.dom.*;
```

Create and open a report.

```
d = Document('myreport', 'docx');
open(d);
```

### Create DOCX Page Header

Get the current page layout object.

```
currentLayout = d.CurrentPageLayout;
```

Create a page header definition for DOCX document.

```
docxheader = DOCXPageHeader();
```

Create a DOM Paragraph object and make it center aligned and bold. Set its font size to 12pt. Append it to the DOCXPageHeader object.

```
p = Paragraph('Sample Traffic Data in Austin');
p.Style = [p.Style, {HAlign('center'), Bold(true), FontSize('12pt')}]};
append(docxheader, p);
```

Assign the created docxheader object to the PageHeaders of the current page layout.

```
currentLayout.PageHeaders = docxheader;
```

### Create Body Content

Create cell arrays for the styles to be used by the formal table and its table entries.

```
dataTableStyle = {Border('solid'), ColSep('solid'), RowSep('solid'), Width('100%')...
    OuterMargin('0pt', '0pt', '0pt', '0pt')};
```

Create some sample data from the Austin traffic camera to include in the table. Then create a FormalTable object and include the sample data in the Header and Body sections.

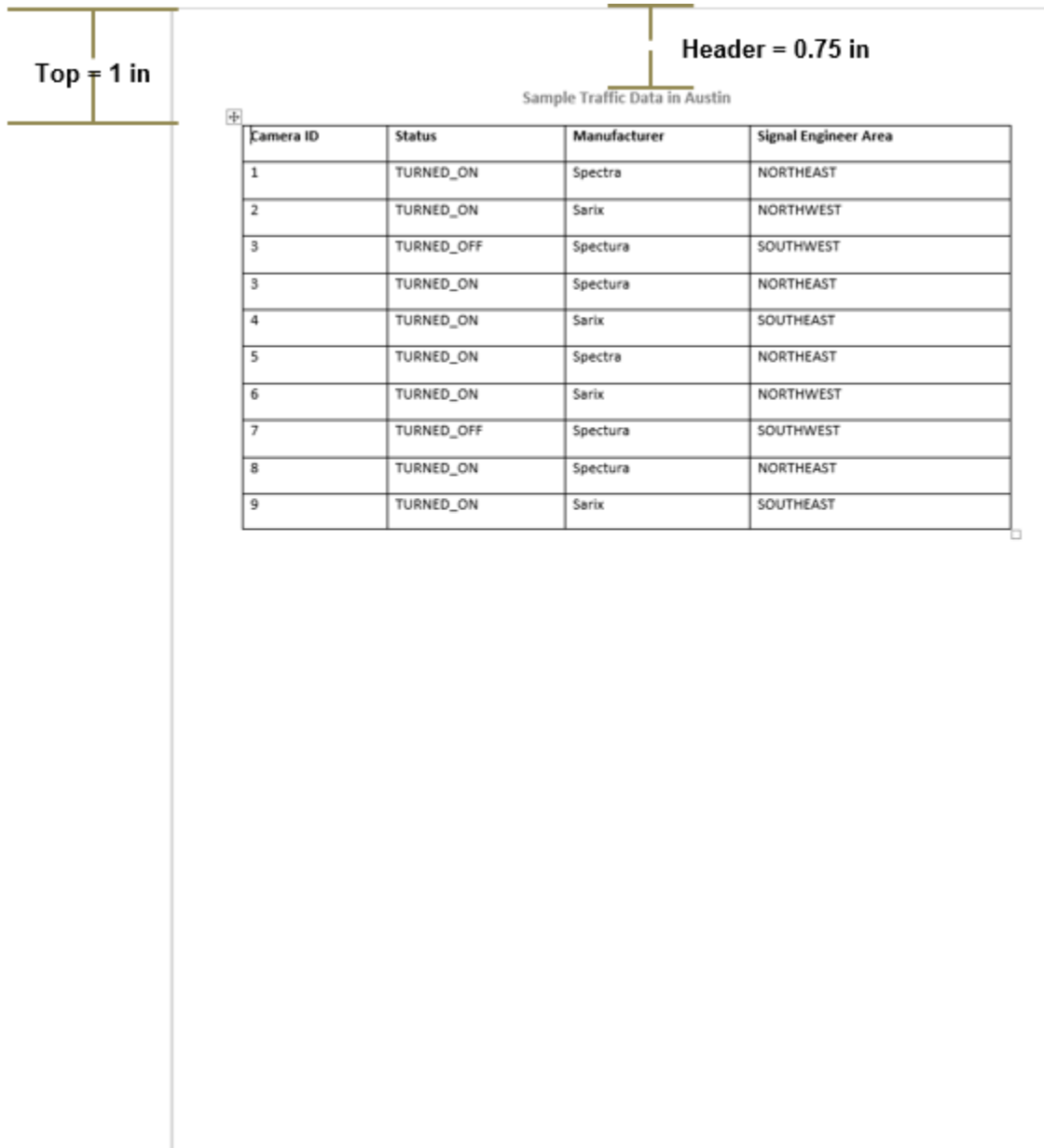
```
dataHeader = {'Camera ID', 'Status', 'Manufacturer', 'Signal Engineer Area'};
dataBody = {'1', 'TURNED_ON', 'Spectra', 'NORTHEAST';
    '2', 'TURNED_ON', 'Sarix', 'NORTHWEST';
    '3', 'TURNED_OFF', 'Spectura', 'SOUTHWEST';
    '3', 'TURNED_ON', 'Spectura', 'NORTHEAST';
    '4', 'TURNED_ON', 'Sarix', 'SOUTHEAST';
    '5', 'TURNED_ON', 'Spectra', 'NORTHEAST';
    '6', 'TURNED_ON', 'Sarix', 'NORTHWEST';
    '7', 'TURNED_OFF', 'Spectura', 'SOUTHWEST';
    '8', 'TURNED_ON', 'Spectura', 'NORTHEAST';
    '9', 'TURNED_ON', 'Sarix', 'SOUTHEAST'};
dataTable = FormalTable(dataHeader, dataBody);
dataTable.Header.Style = [dataTable.Header.Style {Bold}];
dataTable.Style = [dataTable.Style dataTableStyle];
append(d, dataTable);
```

### Set Top Margin and Header Size

The Top property of the PageMargins object specifies the height of the margin. The Header property specifies the distance from the top of the page to the start of the header. The distance from the top of the page to the page body depends on the Top property, Header property and the height of

the header content. For example, if the Header property is less than the Top property, the header starts in the top margin and expands downward to accommodate the header content. The body begins at the bottom of the margin or the header, whichever is greater.

Use the following settings to ensure that the header fits into the Top margin. Set the Top property to 1 inch. 1 inch equals 72 points, so 0.25 inches equals 18 pts. Set the Header value to 0.75 inches as remaining 0.25 inches is enough to accommodate the 12 pts Paragraph created in the DOCX Header.



Top = 1 in

Header = 0.75 in

Sample Traffic Data in Austin

Camera ID	Status	Manufacturer	Signal Engineer Area
1	TURNED_ON	Spectra	NORTHEAST
2	TURNED_ON	Sarix	NORTHWEST
3	TURNED_OFF	Spectura	SOUTHWEST
3	TURNED_ON	Spectura	NORTHEAST
4	TURNED_ON	Sarix	SOUTHEAST
5	TURNED_ON	Spectra	NORTHEAST
6	TURNED_ON	Sarix	NORTHWEST
7	TURNED_OFF	Spectura	SOUTHWEST
8	TURNED_ON	Spectura	NORTHEAST
9	TURNED_ON	Sarix	SOUTHEAST

```
currentLayout.PageMargins.Top = '1in';  
currentLayout.PageMargins.Header = '0.75in';
```



### Create DOCX Page Footer

Create a page footer definition for the DOCX document.

```
docxfooter = DOCXPageFooter();
```

Append a horizontal rule to the docxfooter object.

```
append(docxfooter, HorizontalRule());
```

Append an image to the docxfooter object. Use the DOM ScaleToFit format to scale the image to fit in the page. Assign the created docxfooter object to the PageFooters of the current page layout..

```
imgStyle = {ScaleToFit(true), HAlign('right'), Height('0.30in')};  
img = Image('Logo_footer.Png');  
img.Style = imgStyle;  
append(docxfooter, img);  
currentLayout.PageFooters = docxfooter;
```

### Set Bottom Margin and Footer Size

The **Bottom** property of the **PageMargins** object specifies the height of the page margin. The **Footer** property specifies the distance from the bottom of the page to the bottom of the footer. The distance from the bottom of the page to the page body depends on the settings of the **Bottom** and **Footer** properties and the height of the footer content. For example, if the **Footer** property is less than the **Bottom** property, the footer starts in the bottom margin and expands upward to expand the footer content. The body starts at the top of the margin or the footer, whichever is greater.

Set the **Bottom** property value to 1 inch. To accommodate the 0.30 high **Image** and the horizontal rule created in **DOCX Footer**, set the **Footer** property value to 0.5 inches.

Top = 1 in


Header = 0.75 in

Sample Traffic Data in Austin

Camera ID	Status	Manufacturer	Signal Engineer Area
1	TURNED_ON	Spectra	NORTHEAST
2	TURNED_ON	Serix	NORTHWEST
3	TURNED_OFF	Spectura	SOUTHWEST
3	TURNED_ON	Spectura	NORTHEAST
4	TURNED_ON	Serix	SOUTHEAST
5	TURNED_ON	Spectra	NORTHEAST
6	TURNED_ON	Serix	NORTHWEST
7	TURNED_OFF	Spectura	SOUTHWEST
8	TURNED_ON	Spectura	NORTHEAST
9	TURNED_ON	Serix	SOUTHEAST

Bottom = 1 in

Footer = 0.5 in



```
currentLayout.PageMargins.Bottom = '1in';
currentLayout.PageMargins.Footer = '0.5in';
```

### Set Left Margin, Right Margin and Gutter Size

This example uses the Gutter setting to leave room on the left side of the page for binding the report. The Gutter size is set to 0.25 inches and Left margin is set to 0.5 inches. So, the content starts from 0.75 inches (left margin + gutter) from the left side of the page. The Right margin is set to 0.5 inches.

Top = 1 in

Header = 0.75 in

Sample Traffic Data in Austin


Camera ID	Status	Manufacturer	Signal Engineer Area
1	TURNED_ON	Spectra	NORTHEAST
2	TURNED_ON	Sarix	NORTHWEST
3	TURNED_OFF	Spectra	SOUTHWEST
3	TURNED_ON	Spectra	NORTHEAST
4	TURNED_ON	Sarix	SOUTHEAST
5	TURNED_ON	Spectra	NORTHEAST
6	TURNED_ON	Sarix	NORTHWEST
7	TURNED_OFF	Spectra	SOUTHWEST
8	TURNED_ON	Spectra	NORTHEAST
9	TURNED_ON	Sarix	SOUTHEAST

Left+Gutter = 0.75in

Right = 0.5 in

Bottom = 1 in

Footer = 0.5 in



```
currentLayout.PageMargins.Gutter = '0.25in';
currentLayout.PageMargins.Left = '0.5in';
currentLayout.PageMargins.Right = '0.5in';
```

Generate and display the report.

```
close(d);
rptview(d.OutputPath);
```

## **Set Page Margins in a PDF Report**

This example shows how to define page margins in a PDF report. You can define top, bottom, left, right margins for a PDF page, plus its header, footer and gutter sizes.

**Sample Traffic Data in Austin**

<b>Camera ID</b>	<b>Status</b>	<b>Manufacturer</b>	<b>Signal Engineer Area</b>
1	TURNED_ON	Spectra	NORTHEAST
2	TURNED_ON	Sarix	NORTHWEST
3	TURNED_OFF	Spectura	SOUTHWEST
3	TURNED_ON	Spectura	NORTHEAST
4	TURNED_ON	Sarix	SOUTHEAST
5	TURNED_ON	Spectra	NORTHEAST
6	TURNED_ON	Sarix	NORTHWEST
7	TURNED_OFF	Spectura	SOUTHWEST
8	TURNED_ON	Spectura	NORTHEAST
9	TURNED_ON	Sarix	SOUTHEAST

## Create Report

Import the DOM API packages so you do not have to use long, fully-qualified class names

```
import mlreportgen.dom.*;
```

Create and open a report.

```
d = Document('myreport', 'pdf');
open(d);
```

## Create PDF Page Header

Get the current page layout object.

```
currentLayout = d.CurrentPageLayout;
```

Create a page header definition for the PDF document.

```
pdfheader = PDFPageHeader();
```

Create a DOM Paragraph object and make it center aligned and bold. Set its font size to 12pt. Append it to the PDFPageHeader object.

```
p = Paragraph('Sample Traffic Data in Austin');
p.Style = [p.Style, {HAlign('center'), Bold(true), FontSize('12pt')}]};
append(pdfheader, p);
```

Assign the created pdfheader object to the PageHeaders of the current page layout.

```
currentLayout.PageHeaders = pdfheader;
```

## Create Body Content

Create cell arrays for the styles to be used by the formal table and its table entries.

```
dataTableStyle = {Border('solid'), ColSep('solid'), RowSep('solid'), Width('100%')...
    OuterMargin('0pt', '0pt', '0pt', '0pt')};
```

Create some sample data from the Austin traffic camera to include in the table. Then create a FormalTable object and include the sample data in the Header and Body sections.

```
dataHeader = {'Camera ID', 'Status', 'Manufacturer', 'Signal Engineer Area'};
dataBody = {'1', 'TURNED_ON', 'Spectra', 'NORTHEAST';
    '2', 'TURNED_ON', 'Sarix', 'NORTHWEST';
    '3', 'TURNED_OFF', 'Spectura', 'SOUTHWEST';
    '3', 'TURNED_ON', 'Spectura', 'NORTHEAST';
    '4', 'TURNED_ON', 'Sarix', 'SOUTHEAST';
    '5', 'TURNED_ON', 'Spectra', 'NORTHEAST';
    '6', 'TURNED_ON', 'Sarix', 'NORTHWEST';
    '7', 'TURNED_OFF', 'Spectura', 'SOUTHWEST';
    '8', 'TURNED_ON', 'Spectura', 'NORTHEAST';
    '9', 'TURNED_ON', 'Sarix', 'SOUTHEAST'};
dataTable = FormalTable(dataHeader, dataBody);
dataTable.Header.Style = [dataTable.Header.Style {Bold}];
dataTable.Style = [dataTable.Style dataTableStyle];
append(d, dataTable);
```

## Set Top Margin and Header Size

PDF page headers and footers are fixed in size. The total height from top of the page to the body content is the value of the Top property of the PageMargins object plus the value of the Header property.

Set the Top margin property to 0.75 inches. To accommodate the 12 pts Paragraph created in the PDF Header, set the header size to 0.25 inches. 1 inch equals 72 pts, so 0.25 inches equal 18 pts. The Paragraph content occupies only 12 pts, so 0.25 inches is enough to accommodate the header. The body content starts 1 inch from the top of the page with the header height of 0.25 inches.

Top + Header = 1 in

Sample Traffic Data in Austin

Header = 0.25 in

Camera ID	Status	Manufacturer	Signal Engineer Area
1	TURNED_ON	Spectra	NORTHEAST
2	TURNED_ON	Sarix	NORTHWEST
3	TURNED_OFF	Spectura	SOUTHWEST
3	TURNED_ON	Spectura	NORTHEAST
4	TURNED_ON	Sarix	SOUTHEAST
5	TURNED_ON	Spectra	NORTHEAST
6	TURNED_ON	Sarix	NORTHWEST
7	TURNED_OFF	Spectura	SOUTHWEST
8	TURNED_ON	Spectura	NORTHEAST
9	TURNED_ON	Sarix	SOUTHEAST

```
currentLayout.PageMargins.Top = '0.75in';  
currentLayout.PageMargins.Header = '0.25in';
```

### Create PDF Page Footer

Create a page footer definition for the PDF document.

```
pdffooter = PDFPageFooter();
```

Append a horizontal rule to the `pdffooter` object.

```
append(pdffooter, HorizontalRule());
```

Append an image to the `pdffooter` object. Use the DOM `ScaleToFit` format to scale the image to fit in the page. Assign the created `pdffooter` object to the `PageFooters` of the current page layout.

```
imgStyle = {ScaleToFit(true), HAlign('right'), Height('0.30in')};  
img = Image('Logo.Png');  
img.Style = imgStyle;  
append(pdffooter, img);  
currentLayout.PageFooters = pdffooter;
```

### Set Bottom Margin and Footer Size

The distance from the bottom of the page to the body content is the value of `Bottom` property of the `PageMargins` object plus the value of the `Footer` property.

Set the `Bottom` property value to 0.5 inches. To accommodate the 0.30 in height `Image` and the horizontal rule in the footer, set the `Footer` property value to 0.5 inches. This makes the distance from the bottom of the page to the body content 1 inch.



Top + Header = 1 in


Sample Traffic Data in Austin

Header = 0.25 in

Camera ID	Status	Manufacturer	Signal Engineer Area
1	TURNED_ON	Spectra	NORTHEAST
2	TURNED_ON	Sarix	NORTHWEST
3	TURNED_OFF	Spectura	SOUTHWEST
3	TURNED_ON	Spectura	NORTHEAST
4	TURNED_ON	Sarix	SOUTHEAST
5	TURNED_ON	Spectra	NORTHEAST
6	TURNED_ON	Sarix	NORTHWEST
7	TURNED_OFF	Spectura	SOUTHWEST
8	TURNED_ON	Spectura	NORTHEAST
9	TURNED_ON	Sarix	SOUTHEAST

Bottom + Footer = 1 in

Footer = 0.5 in



```
currentLayout.PageMargins.Bottom = '0.5in';
currentLayout.PageMargins.Footer = '0.5in';
```

### Set Left Margin, Right Margin and Gutter Size

This example uses the `Gutter` setting to leave room on the left side of the page for binding the report. The `Gutter` size is set to 0.25 inches and `Left` margin is set to 0.5 inches. So, the content starts from 0.75 inches (left margin + gutter) from the left side of the page. The `Right` margin is set to 0.5 inches.

Top + Header = 1 in

Header = 0.25 in

Camera ID	Status	Manufacturer	Signal Engineer Area
1	TURNED_ON	Spectra	NORTHEAST
2	TURNED_ON	Sarix	NORTHWEST
3	TURNED_OFF	Spectura	SOUTHWEST
3	TURNED_ON	Spectura	NORTHEAST
4	TURNED_ON	Sarix	SOUTHEAST
5	TURNED_ON	Spectra	NORTHEAST
6	TURNED_ON	Sarix	NORTHWEST
7	TURNED_OFF	Spectura	SOUTHWEST
8	TURNED_ON	Spectura	NORTHEAST
9	TURNED_ON	Sarix	SOUTHEAST

Left + Gutter = 0.75in

Right = 0.5 in

Bottom + Footer = 1 in

Footer = 0.5 in

```
currentLayout.PageMargins.Gutter = '0.25in';
currentLayout.PageMargins.Left = '0.5in';
currentLayout.PageMargins.Right = '0.5in';
```

Generate and display the report.

```
close(d);
rptview(d.OutputPath);
```

## Programmatically Number Pages

These examples show how to number pages of a report programmatically with both the DOM and Report APIs. This workflow applies only to Word and PDF output.

### **Simple Page Numbers Using the DOM API**

In this example, each page contains a footer with the page number in Arabic numerals. Here is the first page:

Hello World

Import the DOM package so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*;
```

Create and open a PDF document. To create a Word document, change the output type from "pdf" to "docx".

```
d = Document("pageNumberExample", "pdf");
open(d);
```

Create a PDF footer and add it to the current page layout. Set the type of the footer to "default" so that it appears on all pages of the document. If the document is a Word document, create a DOCXPageFooter instead of a PDFPageFooter.

```
footer = PDFPageFooter("default");
d.CurrentPageLayout.PageFooters = footer;
```

Define a Paragraph object containing the text appearing before the page number. Format the paragraph so that it is centered horizontally on the page.

```
pagePara = Paragraph("Page");
pagePara.WhiteSpace = "preserve";
pagePara.HAlign = "center";
```

Create a DOM Page object that is a placeholder for a page number. This placeholder will be replaced by the current page number when the PDF document is written, or when the Word document is opened in Word. This Page object can be added directly to a footer or header, but this example includes some extra text and formatting with the page number.

```
pageNum = Page();
```

Append the page number to the paragraph and add the paragraph to the footer.

```
append(pagePara, pageNum);
append(footer, pagePara);
```

Demonstrate the page numbering by adding paragraphs and page breaks to the document to create several pages.

```
page1Para = Paragraph("Hello World");
append(d, page1Para);

append(d, PageBreak());
page2Para = Paragraph("Another page");
append(d, page2Para);

append(d, PageBreak());
page3Para = Paragraph("Another page");
append(d, page3Para);
```

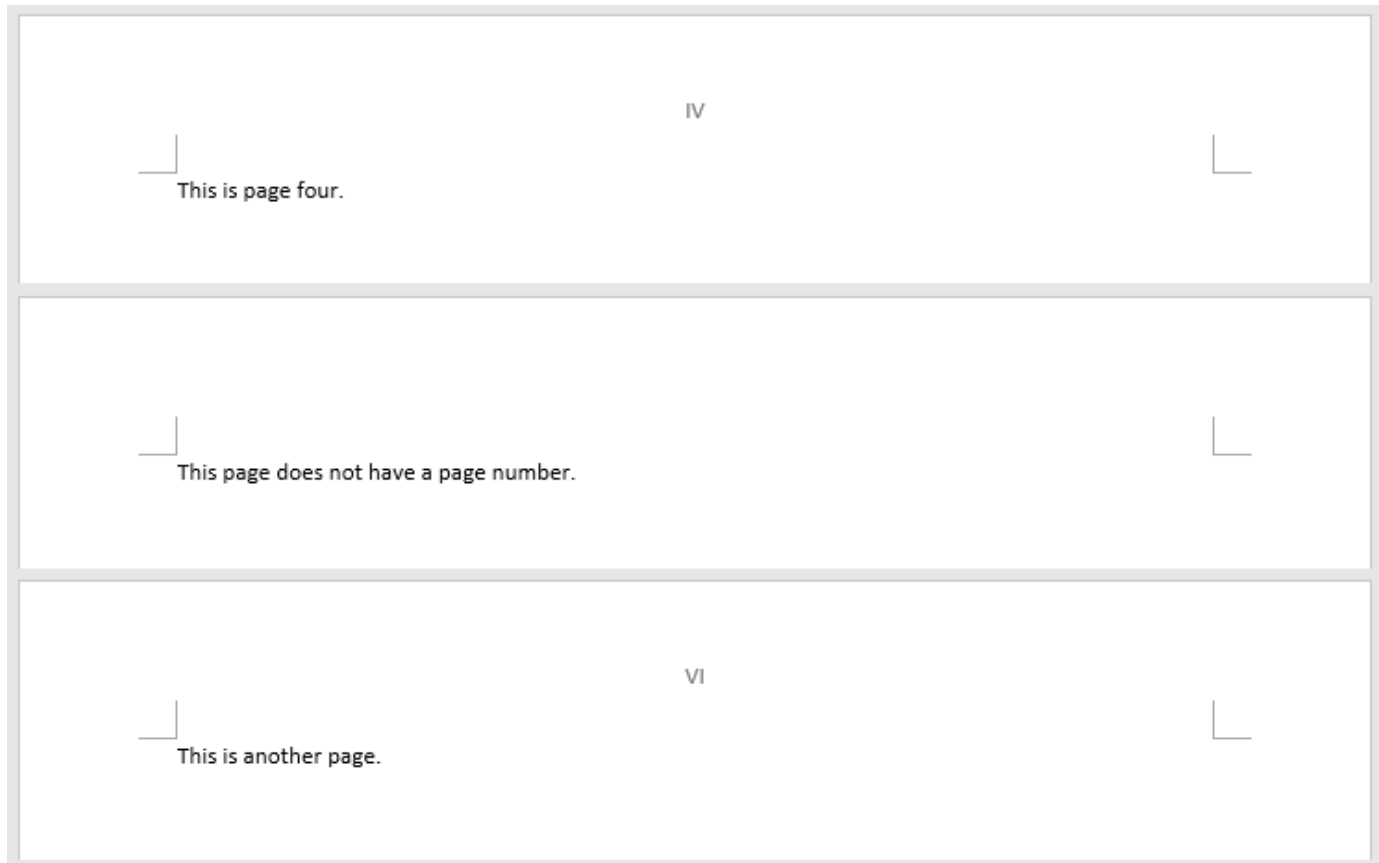
Close and view the document.

```
close(d);
rptview(d);
```

### Customized Page Numbers Using the DOM API

This example demonstrates different ways that you can customize page numbering in a document. A DOM PageNumber object sets the page numbering to begin at 4 and the number format to uppercase

Roman numerals. A different setting in the page header object causes only even pages to have a header. Here are the tops of each page in the document:



Import the DOM package so that you do not have to use long, fully-qualified class names.

```
import mlreportgen.dom.*;
```

Create and open a Word document. To create a PDF document, change the output type from docx to pdf.

```
d = Document("customPageNumberExample", "docx");
open(d);
```

Create a `PageNumber` object to specify that the page numbers start at 4 and use uppercase Roman numerals. Add the object to the current page layout of the document.

```
romanPageNumber = PageNumber(4, "I");

layout = d.CurrentPageLayout;
layout.Style = [layout.Style {romanPageNumber}];
```

Create a header and add it to the current page layout. Set the type of the header to `even` so that it is included only on even pages of the document. If the document is a PDF document, create a `PDFPageHeader` instead of a `DOCXPageHeader`.

```
evenPageHeader = DOCXPageHeader("even");
layout.PageHeaders = evenPageHeader;
```

Create a paragraph containing a page number placeholder. Set the paragraph's `HALign` property so that the page number is centered in the header. Add the paragraph to the header.

```
headerPara = Paragraph();
headerPara.HAlign = "center";
append(headerPara, Page());

append(evenPageHeader, headerPara);
```

Demonstrate the page numbering by adding paragraphs and page breaks to the document to create several pages.

```
page4Para = Paragraph("This is page four.");
append(d, page4Para);

append(d, PageBreak());
page5Para = Paragraph("This page does not have a page number.");
append(d, page5Para);

append(d, PageBreak());
page6Para = Paragraph("This is another page.");
append(d, page6Para);
```

Close and view the document.

```
close(d);
rptview(d);
```

### Page Numbers Using the Report API

Page numbers are automatically included in reports made with the Report API. You can set the starting number and number format for the entire report or for individual chapters. In this example, the default number format for the report is Arabic numerals, but the last chapter overrides this setting to use uppercase alphabetic characters. The table of contents demonstrates the page numbering setup:

## Table of Contents

<a href="#">Chapter 1. Introduction</a> .....	1
<a href="#">1.1. First Section of Chapter 1</a> .....	1
<a href="#">Chapter 2. Next chapter</a> .....	2
<a href="#">2.1. First Section of Chapter 2</a> .....	2
<a href="#">Chapter 3. Last Chapter</a> .....	A
<a href="#">3.1. First Section of the last chapter</a> .....	A
<a href="#">3.2. Second Section of the last chapter</a> .....	B

Import the Report and DOM packages so that you do not have to use long, fully-qualified class names.

```
import mlreportgen.report.*
import mlreportgen.dom.*
```

Create a PDF report. To create a Word report, change the output type from `pdf` to `docx`. You can set the first page number and number format for the entire report by modifying the `PageNumberFormat`

and `FirstPageNumber` properties of the report object's `Layout` property, as shown in the commented lines of code. Because the default page numbering is Arabic numerals beginning with 1, you do not have to set the properties for this example.

```
rpt = Report("newReport", "pdf");  
% rpt.Layout.PageNumberFormat = "n";  
% rpt.Layout.FirstPageNumber = 1;
```

Create a title page and table of contents. Add them to the report. The title page has no page number. The table of contents is numbered by default with lowercase Roman numerals beginning with *i*. To change the table of contents numbering, use the `Layout` property of the object as shown in the commented code for the `Report` object.

```
tp = TitlePage();  
tp.Title = "New Report";  
tp.Author = "MathWorks";  
add(rpt, tp);
```

```
toc = TableOfContents();
```

```
add(rpt, toc);
```

Create two chapters using the default page number and format of the report. Add them to the report.

```
ch = Chapter("Introduction");  
sec = Section("First Section of Chapter 1");  
add(sec, "This is the first section of chapter 1.");  
add(ch, sec);  
add(rpt, ch);
```

```
ch = Chapter("Next chapter");  
sec = Section("First Section of Chapter 2");  
add(sec, "This is the first section of chapter 2.");  
add(ch, sec);  
add(rpt, ch);
```

Create a final chapter for the report. Set the starting page number back to 1 and the number format to uppercase alphabetic characters.

```
ch = Chapter("Last Chapter");  
ch.Layout.FirstPageNumber = 1;  
ch.Layout.PageNumberFormat = "A";
```

Create two sections separated by a page break and add them to the report to demonstrate the page numbering of the final chapter.

```
sec = Section("First Section of the last chapter");  
add(sec, "This is the first section of the last chapter.");  
add(ch, sec);
```

```
add(ch, PageBreak());
```

```
sec = Section("Second Section of the last chapter");  
add(sec, "This is the second section of the last chapter.");  
add(ch, sec);
```

```
add(rpt, ch);
```



Close and view the report.

```
close(rpt);  
rptview(rpt);
```

## Create an Inline Equation in a Report

This example shows how to insert an equation in a line of text in a report. For example:

Here is an inline equation:  $\int_0^2 x^2 \sin(x) dx$

You indicate whether an equation is on a line by itself or in line with the adjacent text by setting the `DisplayInline` property of an equation reporter. If the `DisplayInline` property is set to `false`, the reporter adds an image of the formatted equation on a separate line of a report. If the `DisplayInline` property is set to `true`, you get the image of the formatted equation by calling the `getImpl` method and add the image to a paragraph in the report.

### Import API Packages

Import the DOM and Report API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*
import mlreportgen.dom.*
```

### Create Report

This example creates a single-file HTML report. To create a different type of report, change the output type to "html", "pdf", or "docx". Create a paragraph to contain the equation.

```
rpt = Report("myreport", "html-file");
p = Paragraph("Here is an inline equation: ");
p.FontSize = "14pt";
p.WhiteSpace = "preserve";
```

### Create an Equation Reporter for an Inline Equation

Create an Equation reporter. Specify that the image of the equation is in line with the adjacent text by setting the `DisplayInline` property to `true`.

```
eq = Equation("\int_{0}^{2} x^2\sin(x) dx");
eq.DisplayInline = true;
eq.FontSize = 14;
```

### Add Image of Equation to Report

To get a snapshot image of the formatted equation, call the `getImpl` method. Align the baseline of the equation integrand with the baseline of the text by specifying an amount by which the image is lowered from the baseline of the text. Try different amounts until you are satisfied with the alignment. For HTML and PDF reports, you can specify the amount as a percentage of the line height. For Word reports, specify the amount as a number of units. See the `Value` property of the `mlreportgen.dom.VerticalAlign` class.

```
eqImg = getImpl(eq,rpt);
if (rpt.Type == "html" || rpt.Type == "html-file" || rpt.Type == "pdf")
    eqImg.Style = {VerticalAlign("-30%")};
elseif(rpt.Type == "docx")
```

```
    eqImg.Style = {VerticalAlign("-5pt")};  
end
```

Add the image to the paragraph. Add the paragraph to the report.

```
append(p, eqImg);  
add(rpt, p);
```

### **Close and View Report**

```
close(rpt);  
rptview(rpt);
```

### **See Also**

`mlreportgen.report.Equation` | `mlreportgen.dom.Paragraph` |  
`mlreportgen.dom.VerticalAlign`

### **More About**

- “Report Formatting Approaches” on page 13-17

## Custom Styled Word List

This example shows how to style the following Word multilevel list by defining a new style in a custom Word template.

- 1) a
- 2) b
  - a) 1
    - i) a
    - ii) b
    - iii) c
    - iv) d
  - b) 2
  - c) 3
  - d) 4
- 3) c
- 4) d

### Create Word Template

Import the DOM packages so that you do not have to use the fully qualified class names.

```
import mlreportgen.dom.*
```

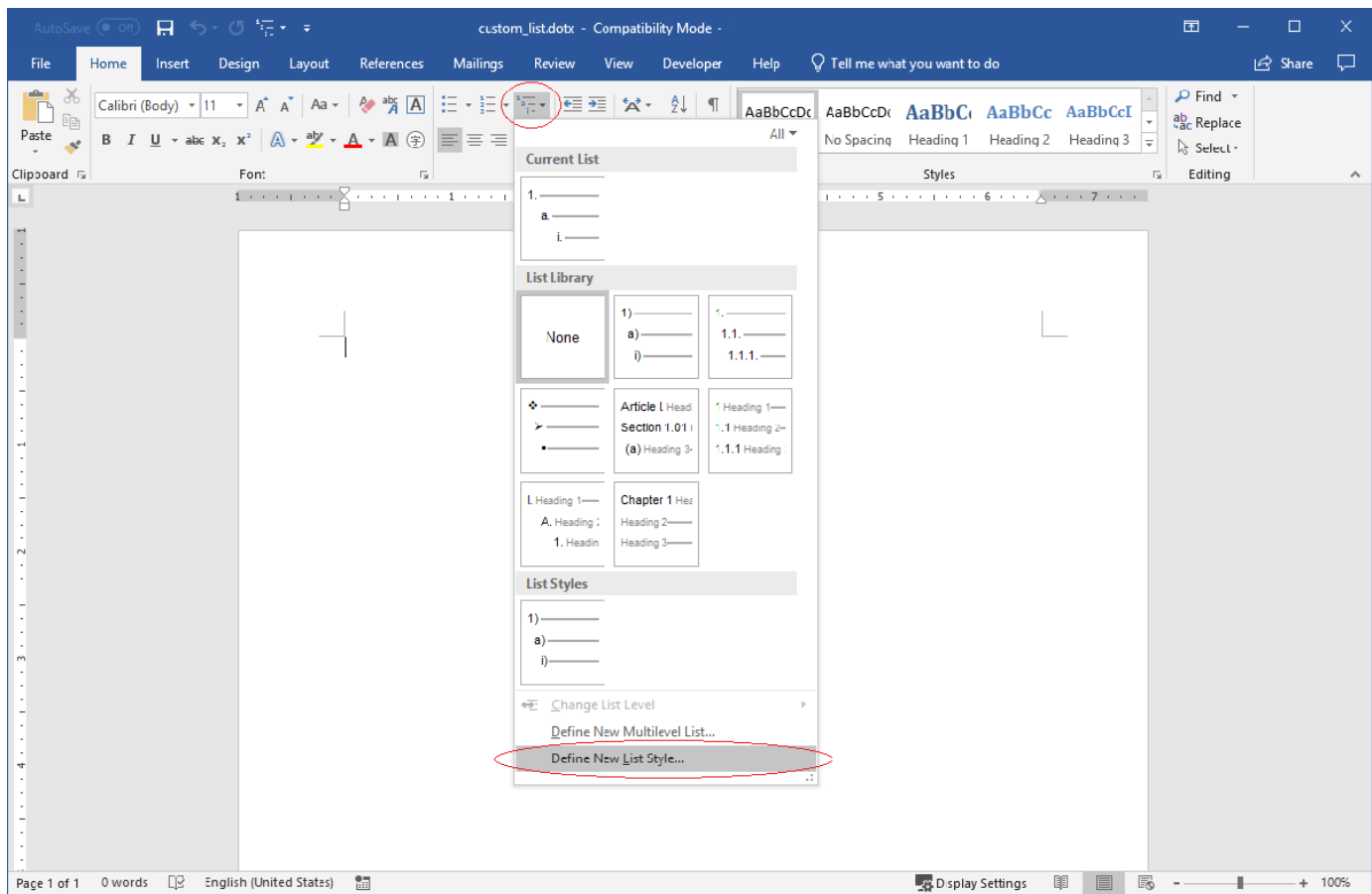
Create a copy of the default Report Generator Word template.

```
Document.createTemplate("custom_list", "docx")
```

```
ans =  
'H:\examples\rptgen-ex80138576\custom_list.dotx'
```

Start the Microsoft Word application and open the `custom_list.dotx` Word template. Do not double-click the `custom_list.dotx` template file. Double-clicking a template file creates a new Word document that is based on the template.

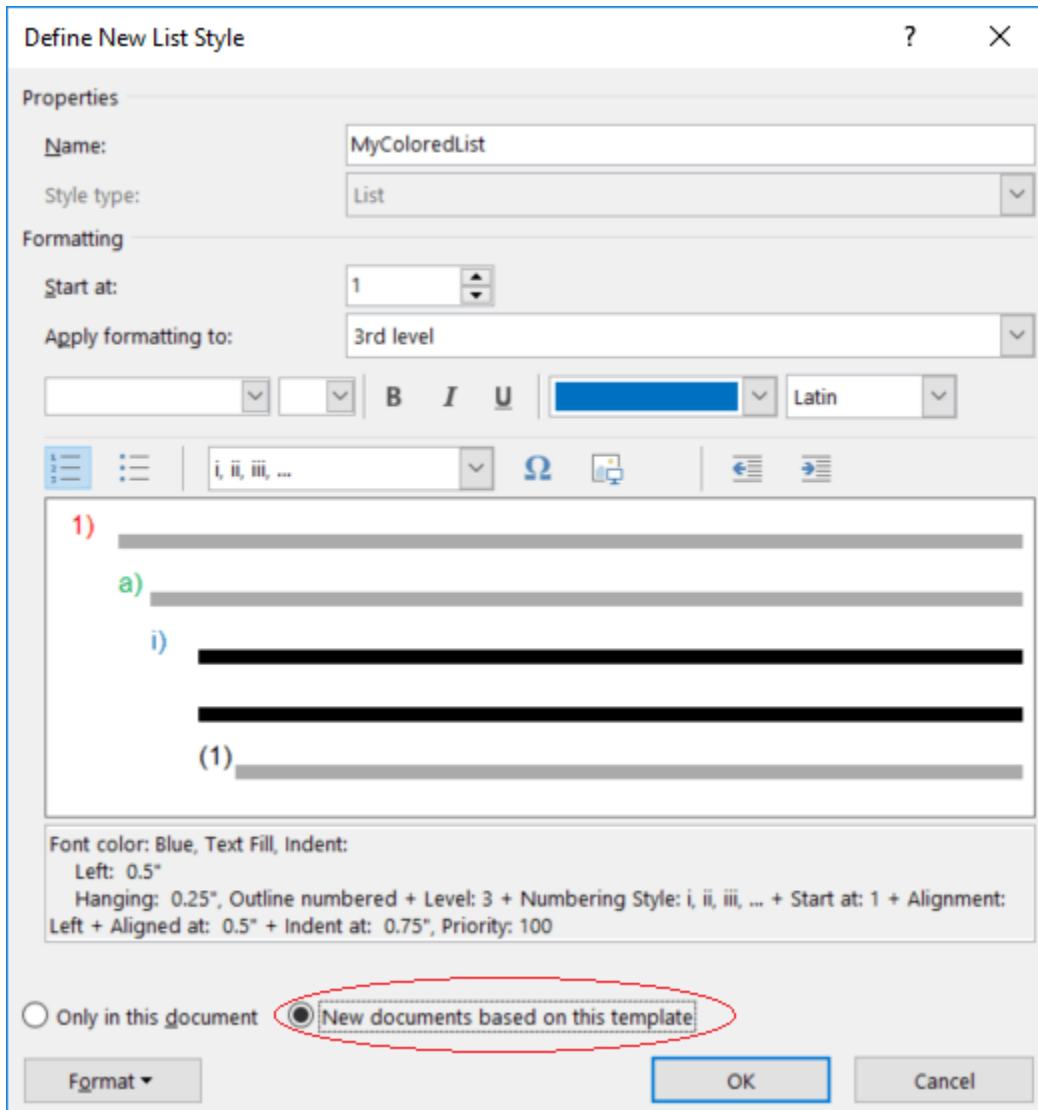
Click the Multilevel List arrow. Then click Define New List Style.



In the **Define New List Style** dialog box, enter a name for the style. For this example, use `MyColoredList` as the name. This name is used by the Report Generator to reference the custom multilevel list style.

Customize the multilevel list. For this example, the colors of first, second, and third levels, are red, green, and blue, respectively.

Before closing the dialog box, select **New documents based on this template**.



Closing the dialog creates a list. Delete this list, otherwise it will become boiler plate text for all new documents.

Save the template.

### Create Document

Create a Word document using the template that you created, `custom_list.dotx`.

The following code uses the template, `custom_list_example.dotx`, that is provided with this example. This template already specifies a colored list and is provided so that the example runs properly. If you are repeating the example steps, replace `custom_list_example.dotx` with `custom_list.dotx`.

```
d = Document("multilevel", "docx", "custom_list_example.dotx");
```

Create a multilevel list by using either an `UnorderedList` or an `OrderedList` object. The type of list DOM object does not matter. The style name controls the appearance of the list.

```
threeLevelCellArray = {
  'a', ...
  'b', ...
  { ...
    '1', ...
    { ...
      'a', ...
      'b', ...
      'c', ...
      'd' ...
    }, ...
    '2', ...
    '3', ...
    '4' ...
  }, ...
  'c', ...
  'd'};
list = UnorderedList(threeLevelCellArray);
list.StyleName = "MyColoredList";
```

Append the list to a Document object.

```
append(d, list);
```

Close and view the document.

```
close(d);
rptview(d);
```

## See Also

[mlreportgen.dom.Document](#) | [mlreportgen.dom.UnorderedList](#)

## More About

- “Create a Microsoft Word Template” on page 13-119
- “Create and Format Lists” on page 13-51
- “Multilevel List” on page 17-86

## Multilevel List

This example shows two ways to create multilevel lists. The first way creates a cell array that models a multilevel list and appends the cell array to the document. The second way programmatically builds up the multilevel list by constructing List and ListItem objects.

- a
- b
  - 1
    - a
    - b
    - c
    - d
  - 2
  - 3
  - 4
- c
- d

### Setup

Import the DOM packages so that you do not have to use the fully qualified class names.

```
import mlreportgen.dom.*
```

### Cell Array List

#### Simple List

A simple list can be represented as a simple cell array where cell array elements are used to create list items. To create this simple list:

- a
- b
- c
- d

Create this cell array.

```
simpleCellArray = { ...  
    'a', ...  
    'b', ...  
    'c', ...  
    'd'};
```



Append the cell array to a Document object. To create an HTML report, replace "pdf" with "html". To create a Word report, replace "pdf" with "docx".

```
d = Document("cell_simple_list", "pdf");
append(d, simpleCellArray);
close(d);
rptview(d);
```

### Two-Level List

A two-level list can be represented as a cell array where one or more elements are cells. To create this two-level list:

- a
- b
  - 1
  - 2
  - 3
  - 4
- c
- d

Create this cell array:

```
twoLevelCellArray = { ...
    'a', ...
    'b', ...
    { ...
        '1', ...
        '2', ...
        '3', ...
        '4' ...
    }, ...
    'c', ...
    'd'};
```

Append the two-level cell array to a Document object. To create an HTML report, replace "pdf" with "html". To create a Word report, replace "pdf" with "docx".

```
d = Document("cell_two_level_list", "pdf");
append(d, twoLevelCellArray);
close(d);
rptview(d);
```

### Three Level List

A three-level list can be represented as a nested cell array that is three levels deep. To create this three-level list:

- a
- b
  - 1
    - a
    - b
    - c
    - d
  - 2
  - 3
  - 4
- c
- d

Create this cell array:

```
threeLevelCellArray = {  
    'a', ...  
    'b', ...  
    { ...  
        '1', ...  
        { ...  
            'a', ...  
            'b', ...  
            'c', ...  
            'd' ...  
        }, ...  
        '2', ...  
        '3', ...  
        '4' ...  
    }, ...  
    'c', ...  
    'd'};
```

Append the three-level cell array to a **Document** object. To create an HTML report, replace "pdf" with "html". To create a Word report, replace "pdf" with "docx".

```
d = Document("cell_three_level_list", "pdf");  
append(d, threeLevelCellArray);  
close(d);  
rptview(d);
```

To create even deeper multi-level lists, add more nested cell arrays to represent inner lists.

## Programmatic List

### Simple List

A simple list can be constructed by creating `ListItem` objects and appending them to an `OrderedList` or `UnorderedList` object. For the following simple unordered list:

- a
- b
- c
- d

Create `ListItem` objects.

```
itemA = ListItem('a');
itemB = ListItem('b');
itemC = ListItem('c');
itemD = ListItem('d');
```

Append the `ListItem` objects to an `UnorderedList` object.

```
unorderedList = UnorderedList();
append(unorderedList, itemA);
append(unorderedList, itemB);
append(unorderedList, itemC);
append(unorderedList, itemD);
```

Append the list to a `Document` object. To create an HTML report, replace "pdf" with "html". To create a Word report, replace "pdf" with "docx".

```
d = Document("prog_simple_list", "pdf");
append(d, unorderedList);
close(d);
rptview(d);
```

### Two-Level List

A two-level list can be constructed by appending either an `OrderedList` or an `UnorderedList` object to the parent `List` Object. For the following two-level list:

1. a
2. b
  - 1
  - 2
  - 3
  - 4
3. c
4. d

Create a second level unordered list by using a cell array. To create an HTML report, replace "pdf" with "html". To create a Word report, replace "pdf" with "docx". For Word report, mixing unordered and ordered lists may not produce the best results. See the custom styled word list example.

```
secondLevelList = UnorderedList({ ...
    '1', ...
    '2', ...
    '3', ...
    '4'});
```

Create first level list.

```
itemA = ListItem('a');
itemB = ListItem('b');
itemC = ListItem('c');
itemD = ListItem('d');

firstLevelList = OrderedList();
append(firstLevelList, itemA);
append(firstLevelList, itemB);
append(firstLevelList, secondLevelList); % Not a ListItem, but an OrderedList
append(firstLevelList, itemC);
append(firstLevelList, itemD);
```

Append the list to a Document object.

```
d = Document("prog_two_level_list", "pdf");
append(d, firstLevelList);
close(d);
rptview(d);
```

### Three-Level List

A three-level list can be constructed by appending a two-level list to the parent List Object. For the following three-level list:

- ```

1. a
2. b
   1. a
   2. b
      • 1
      • 2
      • 3
      • 4
   3. c
   4. d
3. c
4. d
```

Create the third-level list.

```
thirdLevelList = UnorderedList({ ...
    '1', ...
    '2', ...
    '3', ...
    '4'});
```

Create the second-level list.

```
secondLevelList = OrderedList({ ...
    'a', ...
    'b', ...
    thirdLevelList, ... % This is a List, the rest are ListItems.
    'c', ...
    'd'});
```

Create the first-level list.

```
firstLevelList = OrderedList({
    'a', ...
    'b', ...
    secondLevelList, ...
    'c', ...
    'd'});
```

Append the list to a Document object. To create a Word report, replace "pdf" with "docx". To create an HTML report, replace "pdf" with "html". For Word report, mixing unordered and ordered lists may not produce the best results. See the custom styled word list example.

```
d = Document("prog_three_level_list", "pdf");
append(d, firstLevelList);
close(d);
rptview(d);
```

To create even deeper multilevel lists, append List objects to List objects.

## See Also

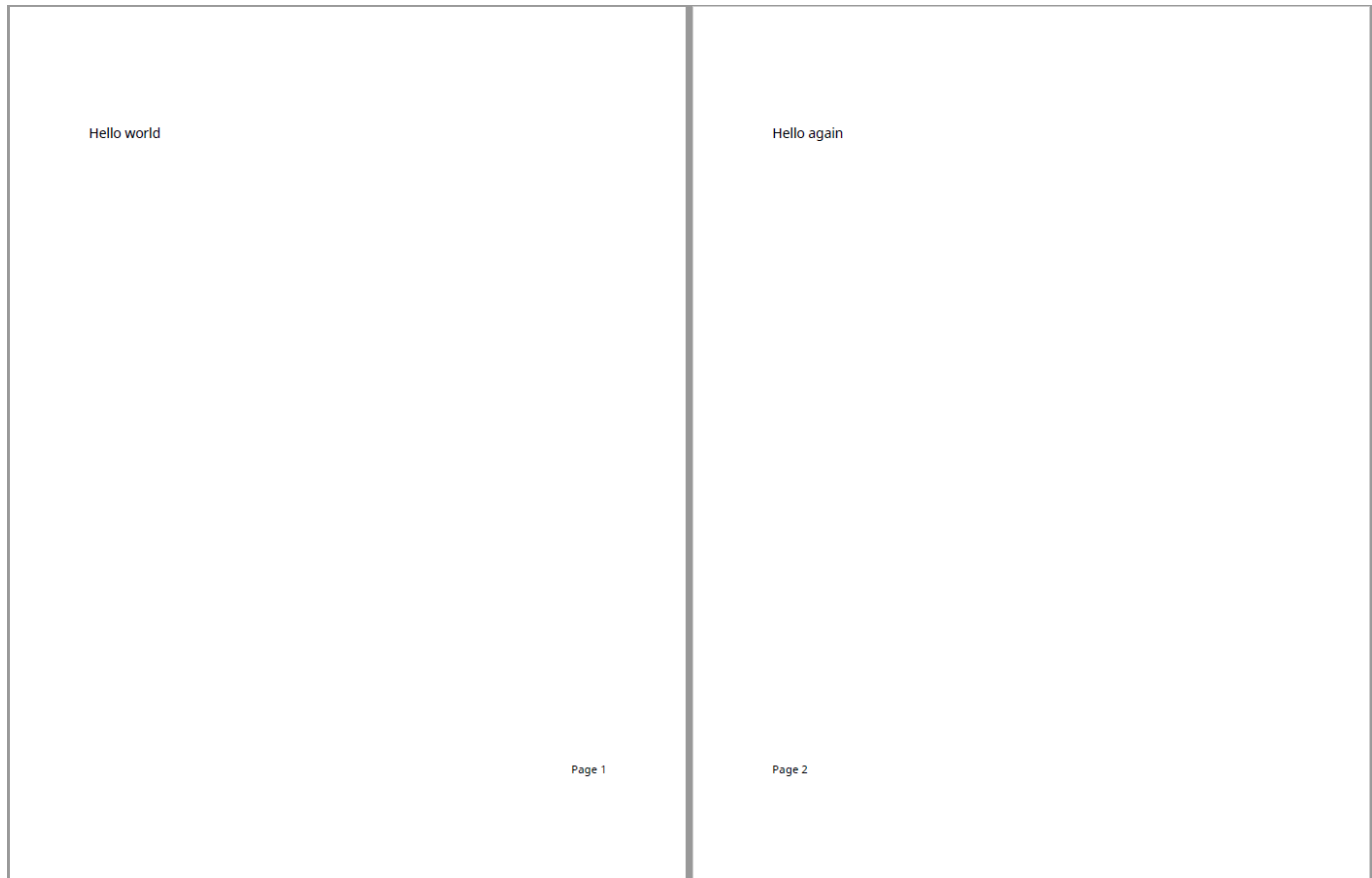
[mlreportgen.dom.Document](#) | [mlreportgen.dom.ListItem](#) |  
[mlreportgen.dom.OrderedList](#) | [mlreportgen.dom.UnorderedList](#)

## More About

- “Create and Format Lists” on page 13-51
- “Custom Styled Word List” on page 17-82

## Number Pages in a PDF Template

This example shows how to number pages of a report using a PDF template. It defines a right-aligned page number for odd pages and a left-aligned page number for even pages by using two different footers. The following image shows two pages in a document created using the example template:



### Create a Template

Create a PDF template, `myPDFTemplate.pdfxtx`, in the current working directory. Unzip the template for editing. A reference template that already includes the footers for the example is available in `exampleTemplate.pdfxtx`.

```
mlreportgen.dom.Document.createTemplate("myPDFTemplate", "pdf");
unzipTemplate("myPDFTemplate.pdfxtx", "myPDFTemplate_pdfxtx");
%unzipTemplate("exampleTemplate.pdfxtx", "exampleTemplate_pdfxtx");
```

### Define Footer Contents

In `myPDFTemplate_pdfxtx\docpart_templates.html`, define template parts that hold the content of the footers. In the `<dplibrary>` tags, create `<dptemplate>` elements named `MyPageFooter` for odd pages and `MyEvenFooter` for even pages. Create paragraphs that contain the text included in each footer as well as page elements where the page number should be placed. Set the `text-align` style to `right` for `MyPageFooter` and `left` for `MyEvenFooter`. For example:

```

<dplibrary>
  <dptemplate name="rgChapter">
    <h1 class="rgChapterTitle">
      <hole id="rgChapterTitlePrefix" default-style-name="rgChapterTitlePrefix" /><span> </span>
      <hole id="rgChapterTitleNumber" default-style-name="rgChapterTitleNumber" /><span>. </span>
      <hole id="rgChapterTitleText" default-style-name="rgChapterTitleText" />
    </h1>
    <hole id="rgChapterContent"/>
  </dptemplate>
  <dptemplate name="ReportTOC"><TOC number-of-levels ="3" leader-pattern="dots" /></dptemplate>

  <!-- Document part templates defining the footers -->
  <dptemplate name="MyPageFooter">
    <p style="text-align:right;font-size:10pt;white-space:preserve">Page <page/></p>
  </dptemplate>

  <dptemplate name="MyEvenFooter">
    <p style="text-align:left;font-size:10pt;white-space:preserve">Page <page/></p>
  </dptemplate>
</dplibrary>

```

### Create Footer Elements

In the body section of myPDFTemplate\_pdftx\root.html, uncomment the <layout> element and add two <pfooter> elements. Set the type and template-name attributes as shown in the following example HTML code. The default type footer is used for first and odd pages. The even type footer is used for even pages. The template-name attributes are set to the names of the template parts defined earlier. To specify the number of the starting page, add a <pnumber> element.

```

<html>
<head>
  <meta charset="utf-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <title>Report Template</title>
  <link rel="StyleSheet" href="./stylesheets/root.css" type="text/css" />
</head>

<body>

  <!-- Uncomment and edit this layout to customize a document or document part layout based on this -->
  <layout style="page-margin: 1in 1in 1in 1in 0.5in 0.5in 0in; page-size: 8.5in 11in portrait">
    <pfooter type="default" template-name="MyPageFooter"/>
    <pfooter type="even" template-name="MyEvenFooter"/>
    <pnumber format="1" />
  </layout>

</body>
</html>

```

### Zip the Template

Zip the template files back to the myPDFTemplate\_pdftx template package.

```
zipTemplate('myPDFTemplate_pdftx', 'myPDFTemplate_pdftx');
```

### Use the Template

Use the template by specifying the template name when creating the document. The following code uses the reference template `exampleTemplate.pdf` to create a document. To use the template modified by the example, replace `exampleTemplate` with `myPDFTemplate`.

```
import mlreportgen.dom.*

d = Document("myDocument", "pdf", "exampleTemplate");
open(d);

append(d, "Hello world");
append(d, PageBreak());
append(d, "Hello again");
append(d, PageBreak());
append(d, "Hello again");
append(d, PageBreak());
append(d, "Hello again");
append(d, PageBreak());
append(d, "Hello again");

close(d);
rptview(d);
```

### See Also

`mlreportgen.dom.Document`

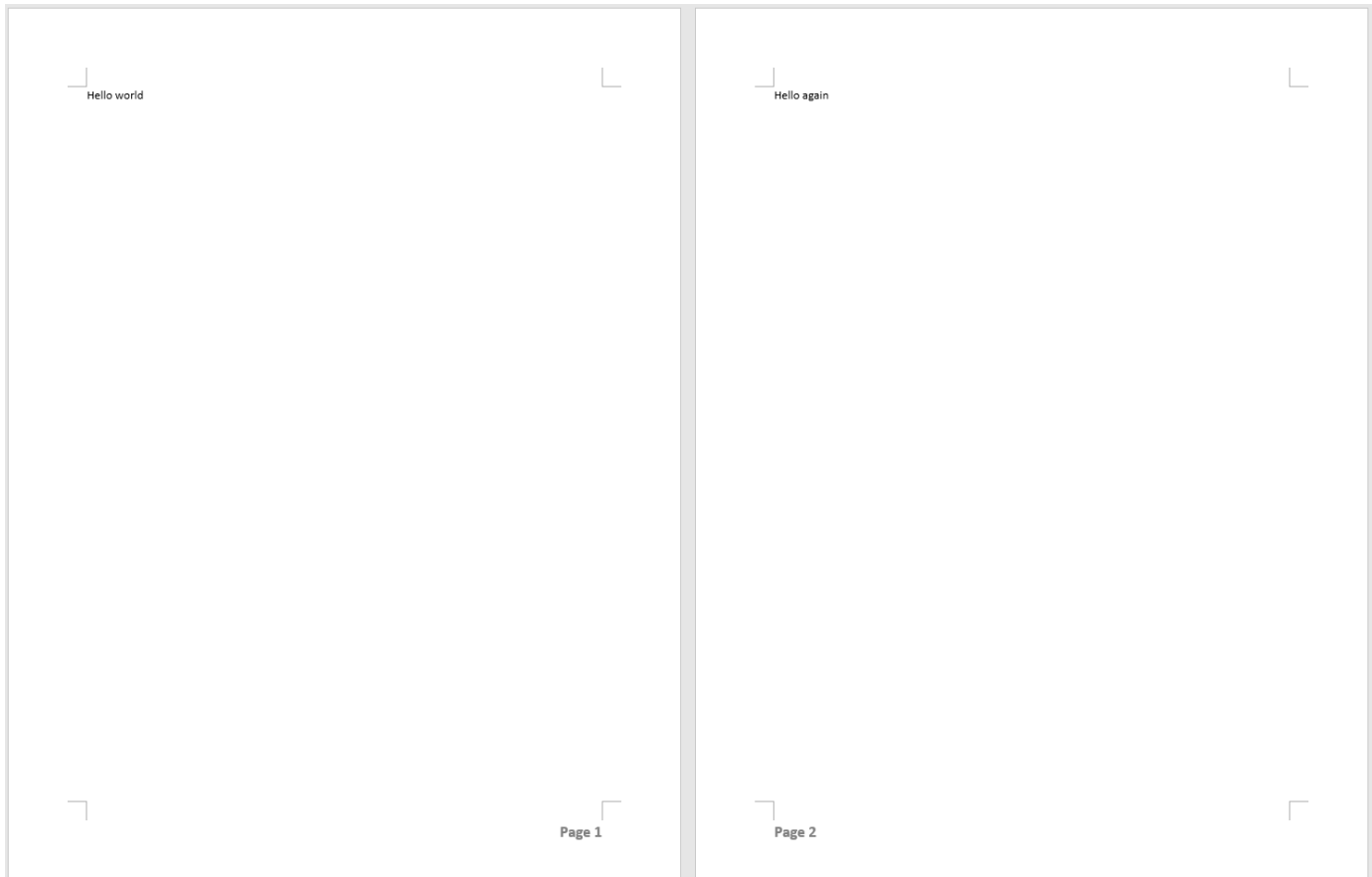
### More About

- “Programmatically Number Pages” on page 17-73
- “Create an HTML or PDF Template” on page 13-130
- “Number Pages in a Word Template” on page 17-95



## Number Pages in a Word Template

This example shows how to number pages of a report using a Word template. It defines a right-aligned page number for odd pages and a left-aligned page number for even pages by using two different footers. The following image shows two pages of a document created using the example template:



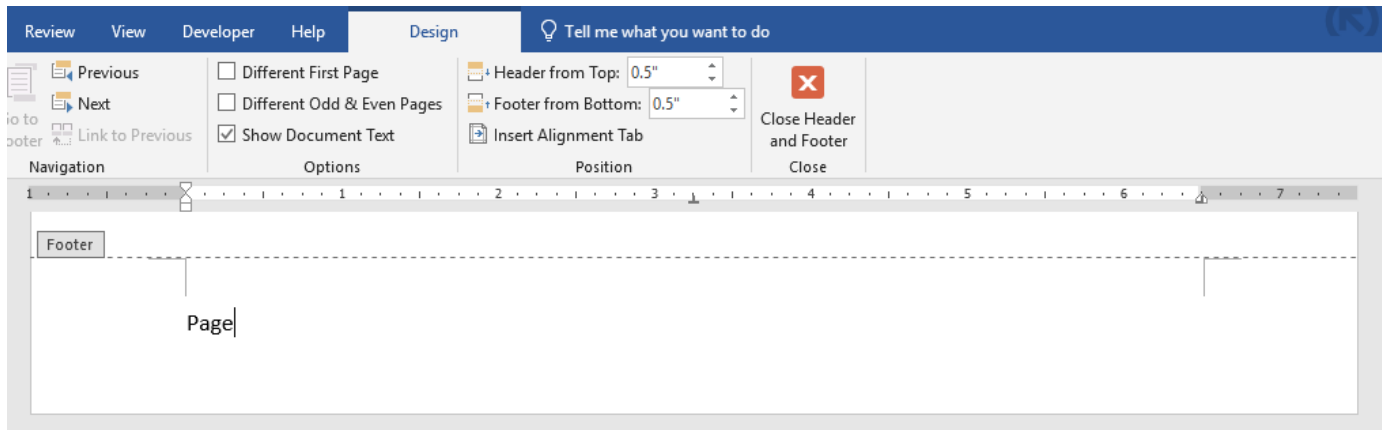
### Create a Template

Create a Word template, `myWordTemplate.dotx`, in the current working directory. Open the file in Word. A reference template that already includes the footers for the example is available in `exampleTemplate.dotx`.

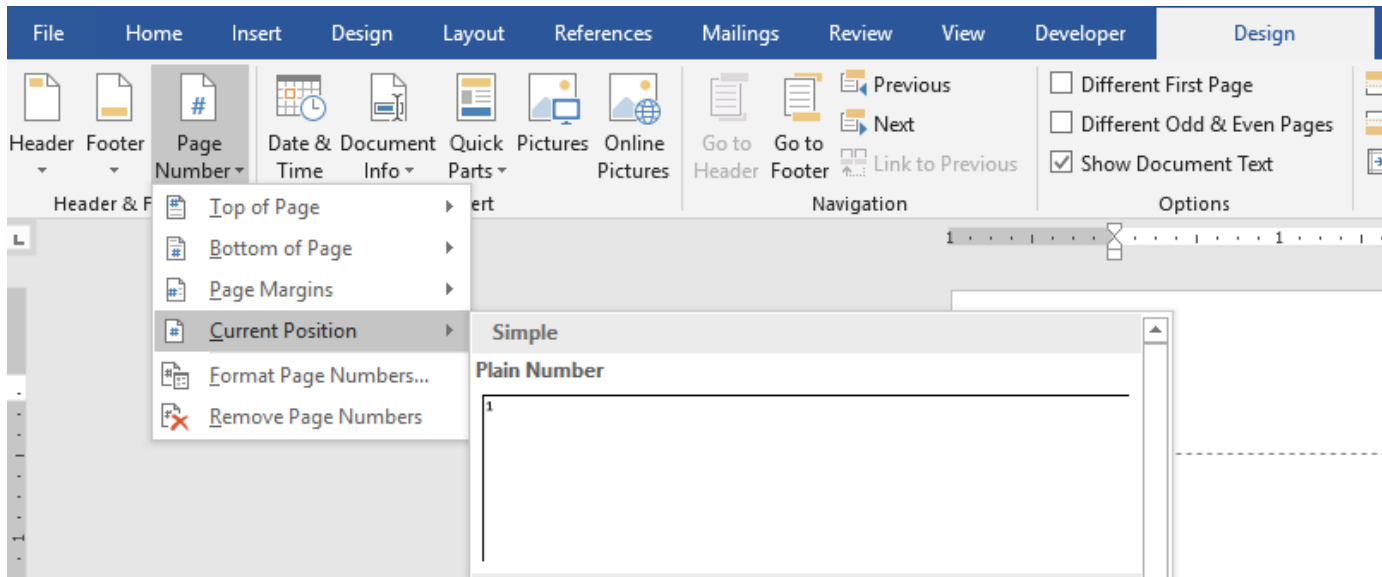
```
mlreportgen.dom.Document.createTemplate("myWordTemplate", "docx");
```

### Add a Footer and Page Number

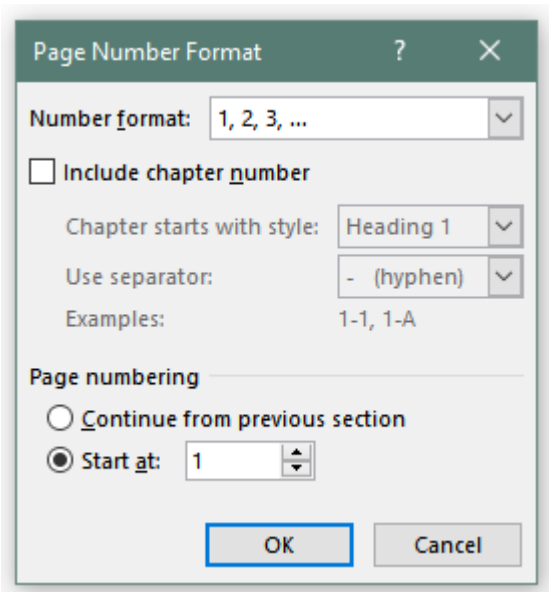
In Word, edit the template footer by double-clicking the footer area. This opens a **Header and Footer Tools Design** tab and moves the cursor into the footer area. Type what should appear before the page number in the footer. In the following example, "Page " precedes the page number:



In the **Header and Footer Tools Design** tab open the **Page Number** menu and select **Plain Number** from the **Current Position** options. This will insert a page number at the current position of the cursor.



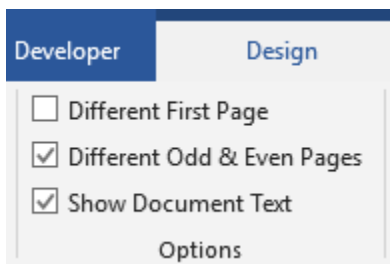
Open the **Page Number** menu again and select **Format Page Numbers...** In the **Page Number Format** menu, set the starting number to 1 and select **OK**.



Format the footer text as you would format any text within Word. In this example, the font size is set to 14 and the text style is bold. Set the horizontal alignment of the text to be right-aligned.

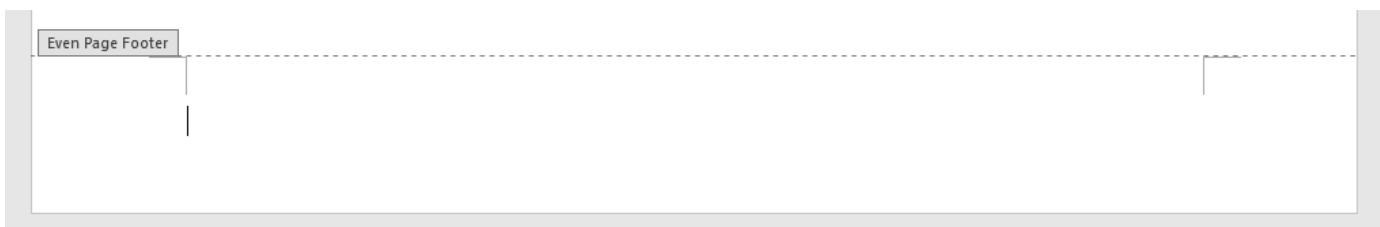
### Create a Different Even Page Footer

To specify a different page footer for even and odd pages, select the **Different Odd & Even Pages** option in the **Header and Footer Tools Design** tab.



To view the even page footer, add another page to the document. Double-click in the body section of the document or click **Close Header and Footer** to edit the main body of the document. Insert a page break by clicking the **Page Break** button in the **Insert** tab or by pressing **CTRL + Enter**.

Double-click the footer area in the new page to edit the even page footer. Repeat the earlier process to add text and a page number to the even page footer. Leave the horizontal alignment of the text as the default left-aligned. There is no need to modify the starting number in the **Page Number Format** menu because the page number is automatically increased from the previous page.



Return to editing the main body of the document. Delete the second page of the document and save the template.

### Use the Template

Use the template by specifying the template name when creating the document. The following code uses the reference template `exampleTemplate.dotx` to create a document. To use the template modified by the example, replace `exampleTemplate` with `myWordTemplate`.

```
import mlreportgen.dom.*

d = Document("myDocument", "docx", "exampleTemplate");
open(d);

append(d, "Hello world");
append(d, PageBreak());
append(d, "Hello again");
append(d, PageBreak());
append(d, "Hello again");
append(d, PageBreak());
append(d, "Hello again");
append(d, PageBreak());
append(d, "Hello again");

close(d);
rptview(d);
```

### See Also

`mlreportgen.dom.Document`

### More About

- “Programmatically Number Pages” on page 17-73
- “Create a Microsoft Word Template” on page 13-119
- “Add Complex Page Numbers in Microsoft Word” on page 13-155
- “Number Pages in a PDF Template” on page 17-92

## Excel to PDF

This example shows how to create a PDF report from a Microsoft® Excel® spreadsheet by using MATLAB® Report Generator™. In this example, the Excel spreadsheet summarizes annual food imports by food category. The source of the spreadsheet is the Economic Research Service, U.S. Department of Agriculture. The spreadsheet is available at U.S. Food Imports. The example uses a local copy of the spreadsheet.

### Import Excel Data

Import the data from the spreadsheet, `Alltables.xlsx`, into MATLAB® cell arrays.

```
xlsfile = "Alltables.xlsx";  
years = readcell(xlsfile, "Sheet", "FOOD$", "Range", "D2:V2");  
data = readcell(xlsfile, "Sheet", "FOOD$", "Range", "D7:V21");  
types = readcell(xlsfile, "Sheet", "FOOD$", "Range", "A7:A21");  
units = readtable(xlsfile, "Sheet", "FOOD$", "Range", "K3:K3", "ReadVariableNames", false);
```

### Create a Report

Import the DOM and Report API packages so that you do not have to use long, fully-qualified class names.

```
import mlreportgen.report.*  
import mlreportgen.dom.*
```

Create a container to hold the report content for a PDF report.

```
rpt = Report("Food Imports Report", "pdf");
```

### Create the Title Page

Create and add a title page to the report.

```
tp = TitlePage();  
tp.Title = "Food Imports Report Based on Multi-Year Data from the USDA";  
tp.Image = "peppers.png";  
tp.Author = "John Doe";  
tp.PubDate = date;  
add(rpt, tp);
```

# Food Imports Report Based on Multi-Year Data from the USDA



**John Doe**

30-Jun-2019

## Create the Table of Contents

Create and add a table of contents by using the `mlreportgen.report.TableofContents` reporter. This reporter automatically creates the table of contents based on the chapter and section titles in the report.

```
toc = TableOfContents();  
add(rpt,toc);
```

| <b>Table of Contents</b>                  |   |
|-------------------------------------------|---|
| <a href="#">1. Imports Summary Graph</a>  | 2 |
| <a href="#">2. Data from 1999 to 2005</a> | 3 |
| <a href="#">3. Data from 2006 to 2012</a> | 4 |
| <a href="#">4. Data from 2013 to 2017</a> | 5 |

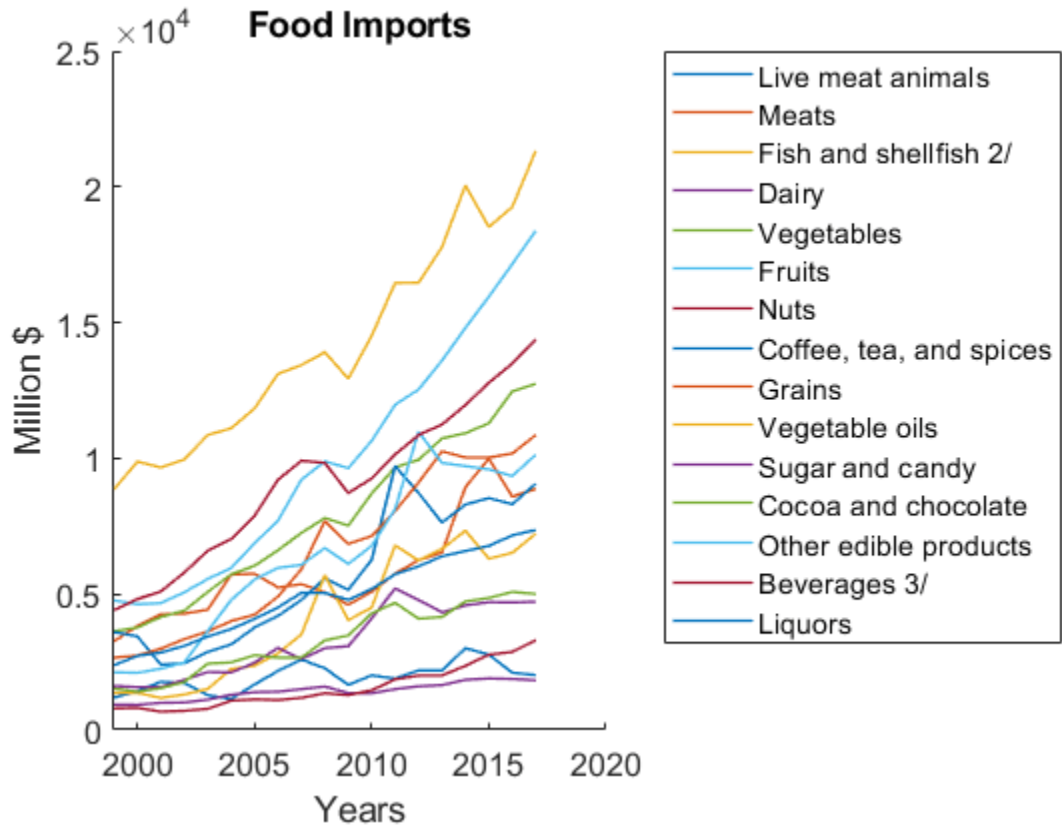
### Plot the Data in MATLAB

Create and format a line plot of the annual imports by food category.

```

fig = figure("Color","w");
ax = axes(fig,"FontSize",12);
t = str2double(years);
for i = 1:size(data,1)
    hold on;
    plot(ax,t,[data{i,:}], "LineWidth",1);
end
xlabel("Years");
ylabel(units.Var1{:});
legend(types,"Location","northeastoutside");
title("Food Imports");

```



**Add the Plot to the Report**

Create a chapter and add the line plot to it.

```
ch = Chapter("Imports Summary Graph");
fig = Figure(fig);
add(ch,fig);
add(rpt,ch);
```

**Convert the Data to a Suitable Form for Creating a DOM Table**

Format the data and create a single string array, tableData, that you can use to create an mlreportgen.dom.Table object.

```
tableData = cellfun(@(x)sprintf("%.0f",x),data);
tableData = [types tableData];
tableHeader = [" " string(years)];
tableData = [tableHeader; tableData]
```

```
tableData = 16x20 string
```

|                         | "1999" | "2000" | "2001" | "2002" | "2003"  | "2004"  | "2005"  |
|-------------------------|--------|--------|--------|--------|---------|---------|---------|
| "Live meat animals"     | "1190" | "1419" | "1771" | "1724" | "1277"  | "1134"  | "1134"  |
| "Meats"                 | "3261" | "3828" | "4256" | "4283" | "4427"  | "5719"  | "5719"  |
| "Fish and shellfish 2/" | "8860" | "9880" | "9663" | "9963" | "10860" | "11106" | "11106" |
| "Dairy"                 | "930"  | "922"  | "996"  | "1009" | "1110"  | "1292"  | "1292"  |
| "Vegetables"            | "3632" | "3771" | "4157" | "4391" | "5082"  | "5730"  | "5730"  |
| "Fruits"                | "4764" | "4629" | "4665" | "5068" | "5558"  | "5962"  | "5962"  |
| "Nuts"                  | "794"  | "809"  | "670"  | "701"  | "776"   | "1078"  | "1078"  |



|                           |        |        |        |        |        |        |
|---------------------------|--------|--------|--------|--------|--------|--------|
| "Coffee, tea, and spices" | "3604" | "3442" | "2401" | "2455" | "2872" | "3144" |
| "Grains"                  | "2659" | "2735" | "2990" | "3343" | "3618" | "4010" |
| "Vegetable oils"          | "1357" | "1362" | "1177" | "1302" | "1507" | "2241" |
| "Sugar and candy"         | "1618" | "1572" | "1581" | "1843" | "2131" | "2111" |
| "Cocoa and chocolate"     | "1522" | "1404" | "1536" | "1761" | "2439" | "2484" |
| "Other edible products"   | "2121" | "2102" | "2252" | "2482" | "3637" | "4784" |
| "Beverages 3/"            | "4412" | "4816" | "5101" | "5795" | "6598" | "7024" |
| "Liquors"                 | "2382" | "2726" | "2847" | "3091" | "3438" | "3709" |

### Create a Food Imports Table in the Report

Create an `mlreportgen.dom.Table` object from the food imports data in the `tableData` variable. Specify the table formats.

```
table = Table(tableData);
table.Style = { ...
    Border("solid"), ...
    RowSep("solid"), ...
    ColSep("solid"), ...
    OuterMargin("5pt", "5pt", "5pt", "5pt")};
table.TableEntriesStyle = {InnerMargin("2pt")};

headerStyle = { ...
    BackgroundColor("LightBlue"), ...
    Bold };

```

```
table.row(1).Style = headerStyle;
```

```
grps = TableColSpecGroup;
grps.Span = 1;
grps.Style = headerStyle;
table.ColSpecGroups = grps;
```

### Fit the Table to the Report Page by Vertically Slicing

To fit a wide table on a report page, divide the table vertically into a set of narrower tables (slices), by using an `mlreportgen.utils.TableSlicer` object.

```
slicer = mlreportgen.utils.TableSlicer(...
    "Table", table, ...
    "MaxCols", 8, ...
    "RepeatCols", 1);
slices = slicer.slice();
```

### Add Chapters for the Table Slices

Create a chapter for each table slice and add the chapters to the report.

```
for slice = slices
    ch = Chapter();
    ch.Title = strjoin(["Data from" years(slice.StartCol-1)...
        "to" years(slice.EndCol-1)]);
    add(ch, slice.Table);
    add(rpt, ch);
end
```

Chapter 2. Data from 1999 to 2005

|                         | 1999 | 2000 | 2001 | 2002 | 2003  | 2004  | 2005  |
|-------------------------|------|------|------|------|-------|-------|-------|
| Live meat animals       | 1190 | 1419 | 1771 | 1724 | 1277  | 1134  | 1672  |
| Meats                   | 3261 | 3828 | 4256 | 4283 | 4427  | 5719  | 5752  |
| Fish and shellfish 2/   | 8860 | 9880 | 9663 | 9963 | 10860 | 11106 | 11840 |
| Dairy                   | 930  | 922  | 996  | 1009 | 1110  | 1292  | 1388  |
| Vegetables              | 3632 | 3771 | 4157 | 4391 | 5082  | 5730  | 6043  |
| Fruits                  | 4764 | 4629 | 4665 | 5068 | 5558  | 5962  | 6874  |
| Nuts                    | 794  | 809  | 670  | 701  | 776   | 1078  | 1122  |
| Coffee, tea, and spices | 3604 | 3442 | 2401 | 2455 | 2872  | 3144  | 3771  |
| Grains                  | 2659 | 2735 | 2990 | 3343 | 3618  | 4010  | 4241  |
| Vegetable oils          | 1357 | 1362 | 1177 | 1302 | 1507  | 2241  | 2363  |
| Sugar and candy         | 1618 | 1572 | 1581 | 1843 | 2131  | 2111  | 2474  |
| Cocoa and chocolate     | 1522 | 1404 | 1536 | 1761 | 2439  | 2484  | 2751  |
| Other edible products   | 2121 | 2102 | 2252 | 2482 | 3637  | 4784  | 5536  |
| Beverages 3/            | 4412 | 4816 | 5101 | 5795 | 6598  | 7024  | 7887  |
| Liquors                 | 2382 | 2726 | 2847 | 3091 | 3438  | 3709  | 4090  |

2

Chapter 3. Data from 2006 to 2012

|                         | 2006  | 2007  | 2008  | 2009  | 2010  | 2011  | 2012  |
|-------------------------|-------|-------|-------|-------|-------|-------|-------|
| Live meat animals       | 2172  | 2588  | 2266  | 1656  | 2007  | 1886  | 2192  |
| Meats                   | 5244  | 5367  | 5060  | 4612  | 5088  | 5755  | 6245  |
| Fish and shellfish 2/   | 13112 | 13435 | 13912 | 12934 | 14517 | 16459 | 16468 |
| Dairy                   | 1406  | 1501  | 1596  | 1353  | 1347  | 1502  | 1604  |
| Vegetables              | 6619  | 7256  | 7801  | 7525  | 8706  | 9667  | 9946  |
| Fruits                  | 7707  | 9217  | 9888  | 9640  | 10649 | 11974 | 12538 |
| Nuts                    | 1099  | 1181  | 1351  | 1278  | 1462  | 1863  | 2000  |
| Coffee, tea, and spices | 4195  | 4791  | 5581  | 5160  | 6265  | 9716  | 8726  |
| Grains                  | 4910  | 5915  | 7690  | 6846  | 7138  | 8063  | 9111  |
| Vegetable oils          | 2818  | 3517  | 5695  | 4039  | 4509  | 6794  | 6236  |
| Sugar and candy         | 3021  | 2606  | 3011  | 3081  | 4107  | 5207  | 4784  |
| Cocoa and chocolate     | 2659  | 2662  | 3299  | 3476  | 4295  | 4681  | 4096  |
| Other edible products   | 5963  | 6080  | 6701  | 6111  | 6782  | 8133  | 10978 |
| Beverages 3/            | 9212  | 9913  | 9833  | 8721  | 9263  | 10143 | 10858 |
| Liquors                 | 4512  | 5048  | 5040  | 4787  | 5189  | 5734  | 6023  |

3

Chapter 4. Data from 2013 to 2017

|                         | 2013  | 2014  | 2015  | 2016  | 2017  |
|-------------------------|-------|-------|-------|-------|-------|
| Live meat animals       | 2190  | 3009  | 2773  | 2103  | 2020  |
| Meats                   | 6530  | 8940  | 9992  | 8589  | 8878  |
| Fish and shellfish 2/   | 17784 | 20054 | 18521 | 19261 | 21324 |
| Dairy                   | 1649  | 1844  | 1893  | 1876  | 1827  |
| Vegetables              | 10734 | 10930 | 11290 | 12469 | 12743 |
| Fruits                  | 13602 | 14808 | 15955 | 17157 | 18383 |
| Nuts                    | 2000  | 2361  | 2767  | 2871  | 3301  |
| Coffee, tea, and spices | 7633  | 8299  | 8536  | 8311  | 9069  |
| Grains                  | 10257 | 10031 | 10030 | 10179 | 10861 |
| Vegetable oils          | 6665  | 7337  | 6304  | 6522  | 7225  |
| Sugar and candy         | 4327  | 4582  | 4705  | 4693  | 4720  |
| Cocoa and chocolate     | 4159  | 4728  | 4860  | 5081  | 5007  |
| Other edible products   | 9831  | 9716  | 9592  | 9351  | 10134 |
| Beverages 3/            | 11248 | 11964 | 12786 | 13493 | 14383 |
| Liquors                 | 6392  | 6587  | 6774  | 7165  | 7363  |

4

**Close and View the Report**

```
close(rpt);  
rptview(rpt)
```

**See Also**

[mlreportgen.dom.Table](#) | [mlreportgen.report.Figure](#) |  
[mlreportgen.utils.TableSlicer](#) | [mlreportgen.report.Section](#) |  
[mlreportgen.report.TitlePage](#) | [mlreportgen.report.TableOfContents](#)

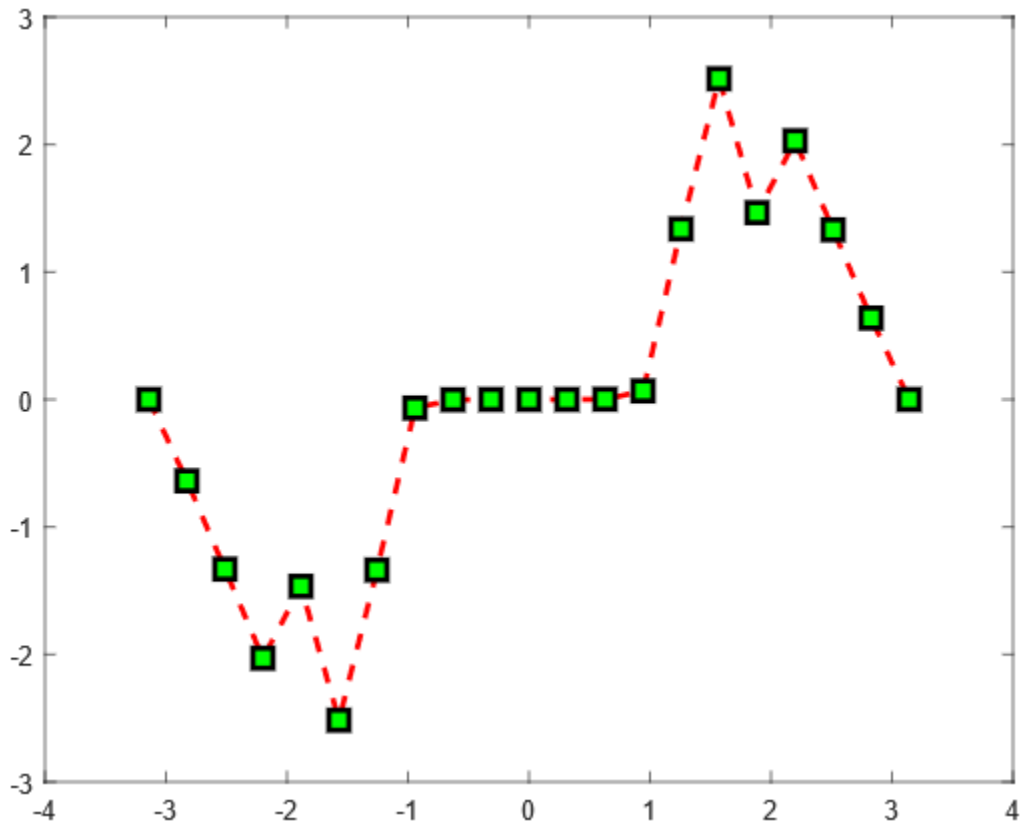
**More About**

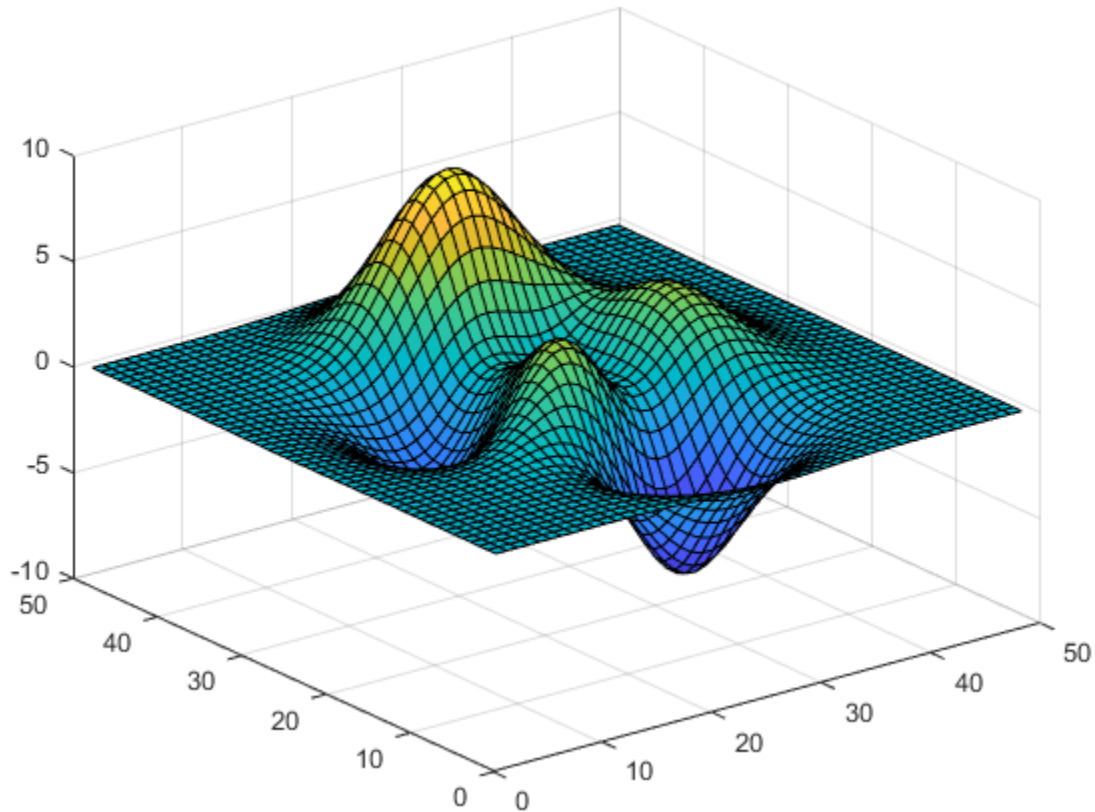
- “Create a Report Program” on page 13-3
- “What Is a Reporter?” on page 1-3

## Prevent MATLAB Figure Display During Report Generation

This example shows how to prevent the display of MATLAB® figures in MATLAB during report generation. If you generate a report that includes several MATLAB figures, you can avoid the overhead of displaying the figures as you create them.

The example creates and includes these MATLAB figures in a report. When the figures are created in MATLAB, the display of the figures is suppressed.





Import the Report API package so that you do not have to use long, fully-qualified class names.

```
import mlreportgen.report.*
```

Create a Word report. You can run this example with other report types by changing the output type.

- To create a single-file HTML report, change the output type to 'html-file'.
- To create a multi-file HTML report, change the output type to 'html'.
- To create a PDF report, change the output type to 'pdf'.

```
rpt = Report('InvisibleFigure','docx');
```

Add a title page and table of contents to the report.

```
add(rpt,TitlePage('Title','Display Invisible Figures','Author','John Doe'));
add(rpt,TableOfContents);
```

Create a chapter and add a figure to it. To prevent the display of the figure in MATLAB, set the `Visible` property of the figure to 'off'.

```
ch = Chapter('Invisible Figure 1');
x = -pi:pi/10:pi;
y = tan(sin(x)) - sin(tan(x));
f1 = figure('visible','off');
plot(x,y,'--rs','LineWidth',2,...
      'MarkerEdgeColor','k',...
      'MarkerFaceColor','g',...);
```

```
        'MarkerSize',10)  
add(ch,Figure(f1));  
add(rpt,ch);
```

Create a second chapter and add an invisible figure to it.

```
ch = Chapter('Invisible Figure 2');  
f2 = figure('visible','off');  
surf(peaks);  
add(ch,Figure(f2));  
add(rpt,ch);
```

Close and view the report.

```
close(rpt);  
rptview(rpt);
```

### See Also

[mlreportgen.report.Figure](#) | [Figure Properties](#) | [mlreportgen.report.Report](#) |  
[mlreportgen.report.Chapter](#) | [mlreportgen.report.TableOfContents](#) |  
[mlreportgen.report.TitlePage](#)

### More About

- “What Is a Reporter?” on page 1-3

## Create a Table from a Cell Array

This example shows how to use a cell array to create a table that displays data of several different data types.

| Book                                          | Year Published | Characters                                                                                                           |
|-----------------------------------------------|----------------|----------------------------------------------------------------------------------------------------------------------|
| <a href="#">The Three Musketeers</a>          | 1844           | <ul style="list-style-type: none"> <li>• d'Artagnan</li> <li>• Athos</li> <li>• Porthos</li> <li>• Aramis</li> </ul> |
| <a href="#">Little Women</a>                  | 1868           | <ul style="list-style-type: none"> <li>• Meg</li> <li>• Jo</li> <li>• Beth</li> <li>• Amy</li> </ul>                 |
| <a href="#">The Hound of the Baskervilles</a> | 1902           | <ul style="list-style-type: none"> <li>• Sherlock</li> <li>• Watson</li> </ul>                                       |

### Document and Style Setup

The following code sets up a document and table styles to be used to format the example tables.

Import the DOM package so you do not have to use long, fully-qualified class names.

```
import mlreportgen.dom.*
```

Create and open a document. To create a PDF document, change the output type from docx to pdf. To create an HTML document, change docx to html or html - file for a multi-file or single-file document, respectively.

```
d = Document("mydoc", "docx");
open(d);
```

Define styles for the table and header section of the table.

```
tableStyles = { ColSep("solid"), ...
               RowSep("solid"), ...
               Border("solid") };

tableHeaderStyles = { BackgroundColor("lightgray"), ...
                    Bold(true) };
```

### Table Creation

Use a cell array in a table constructor to create a table from data with more than one data type. The cell array can contain doubles, strings, character vectors, DOM objects, and other arrays. This

example uses a formal table, which is a table that has a body section and optional header and footer sections.

Define labels to display in the table header. The header labels are strings, so they are stored in a string array instead of a cell array.

```
headerLabels = ["Book", "Year Published", "Characters"];
```

Define data to display in the table body. The table data includes external links, numbers, and string arrays, so it is contained in a cell array.

```
book1 = ExternalLink("https://en.wikipedia.org/wiki/The_Three_Musketeers", ...  
                    "The Three Musketeers");  
book2 = ExternalLink("https://en.wikipedia.org/wiki/Little_Women", ...  
                    "Little Women");  
book3 = ExternalLink("https://en.wikipedia.org/wiki/The_Hound_of_the_Baskervilles", ...  
                    "The Hound of the Baskervilles");  
  
books = {book1; book2; book3};  
  
yearPublished = {1844; 1868; 1902};  
  
characters = { ["d'Artagnan", "Athos", "Porthos", "Aramis"]; ...  
              ["Meg", "Jo", "Beth", "Amy"]; ...  
              ["Sherlock", "Watson"] };
```

```
tableData = [books, yearPublished, characters]
```

```
tableData=3x3 cell array  
  {1x1 mlreportgen.dom.ExternalLink}    {[1844]}    {1x4 string}  
  {1x1 mlreportgen.dom.ExternalLink}    {[1868]}    {1x4 string}  
  {1x1 mlreportgen.dom.ExternalLink}    {[1902]}    {1x2 string}
```

Create a table containing the header labels and table data. The single-row string arrays in the table data are converted to DOM `UnorderedList` objects when the table is constructed.

```
cellTbl = FormalTable(headerLabels,tableData);
```

Set the style of the table and table header to the previously defined styles. Set the inner margin of the table entries so that table entry content is separated from the table entry borders by a space of two points. Append the table to the document.

```
cellTbl.Style = [cellTbl.Style, tableStyles];  
cellTbl.Header.Style = [cellTbl.Header.Style, tableHeaderStyles];  
cellTbl.TableEntriesInnerMargin = "2pt";  
append(d,cellTbl);
```

Close and view the document.

```
close(d);  
rptview(d);
```

## See Also

`mlreportgen.dom.FormalTable` | `mlreportgen.dom.ExternalLink`



## **More About**

- “Create and Format Tables” on page 14-69
- “What Is a Cell Array?”

## Format Numbers in a Table

This example shows how to format numbers in a table that is generated by a report generation program. The example creates a table of uniformly distributed random numbers that have a precision of three digits after the decimal point.

|       |       |       |       |
|-------|-------|-------|-------|
| 0.417 | 0.147 | 0.397 | 0.204 |
| 0.720 | 0.092 | 0.539 | 0.878 |
| 0.000 | 0.186 | 0.419 | 0.027 |
| 0.302 | 0.346 | 0.685 | 0.670 |

The numbers are formatted by using an `mlreportgen.dom.NumberFormat` style object with the table.

### Generate Random Numbers

Generate a 4-by-4 array of random numbers. Initialize the random number generator using a seed of 1, so that each time the example runs, `rand` produces the same numbers.

```
format long
rng("default");
rng(1);
randNumbers = rand(4)
```

```
randNumbers = 4x4
```

```
0.417022004702574    0.146755890817113    0.396767474230670    0.204452249731517
0.720324493442158    0.092338594768798    0.538816734003357    0.878117436390945
0.000114374817345    0.186260211377671    0.419194514403295    0.027387593197926
0.302332572631840    0.345560727043048    0.685219500396759    0.670467510178402
```

The numbers display with a precision of 15 digits after the decimal point.

### Create a Document

Import the DOM package so that you do not have to use long, fully-qualified class names.

```
import mlreportgen.dom.*
```

Create a PDF document. To create a Microsoft® Word, HTML, or single-file HTML document, change "pdf" to "docx", "html", or "html-file", respectively.

```
d = Document("randomnumbers", "pdf");
```

### Create a Table

Create a DOM table from the array of random numbers.

```
t = Table(randNumbers);
```

## Specify the Table Style

Specify that numbers in the table have a precision of 3 digits after the decimal point by using an `m\reportgen.dom.NumberFormat` object. Specify the table width, border, and column and row separators.

```
t.Style = [t.Style
    {NumberFormat("%1.3f"),...
    Width("100%"),...
    Border("solid"),...
    ColSep("solid"),...
    RowSep("solid")}];
```

Center the table entries in the table cells.

```
t.TableEntriesHAlign = "center";
```

## Generate the Report

Append the table to the document. Close and view the document.

```
append(d,t);
close(d);
rptview(d);
```

## See Also

[sprintf](#) | [m\reportgen.dom.NumberFormat](#) | [m\reportgen.dom.Table](#)

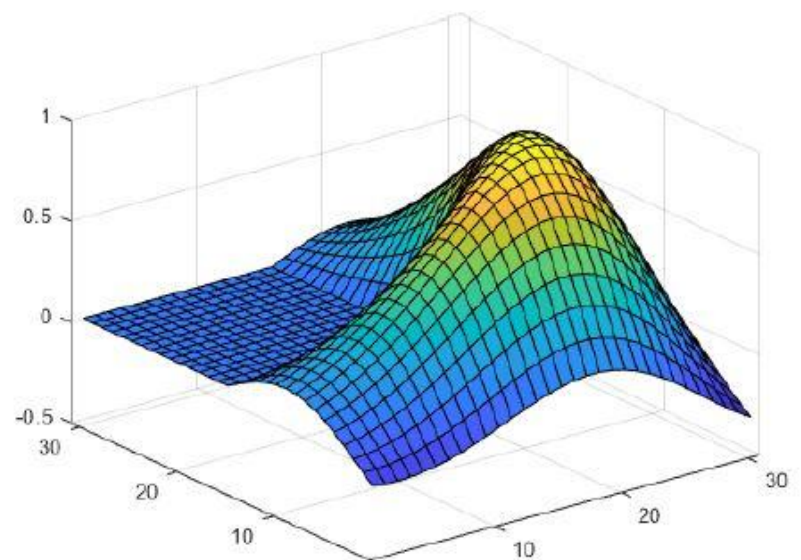
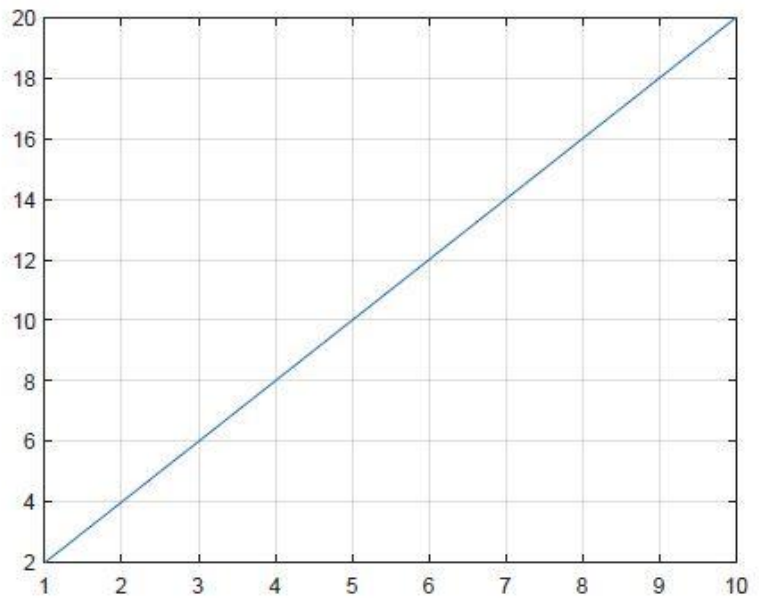
## Related Examples

- “Format Numbers” on page 13-43

## Center Figure Snapshot on a Page

This example shows how to center a figure snapshot on a landscape page for PDF and Microsoft® Word reports.

The example creates a Report API report having landscape layout, creates MATLAB figures, and then uses the `centerFigure` local function to create and add the snapshots of these figures at the center of the page.



## Create Report

Import the DOM and Report API packages so that you do not have to use long, fully-qualified class names.

```
import mlreportgen.dom.*
import mlreportgen.report.*
```

Create a PDF report. To create a Microsoft® Word report, change "pdf" to "docx".

```
rpt = Report("myreport", "pdf");
open(rpt);
```

## Update Report Page Layout

Create a page layout object.

```
if strcmpi(rpt.Type, "pdf")
    pageLayoutObj = PDFPageLayout;
else
    pageLayoutObj = DOCXPageLayout;
end
```

Specify the page orientation, height, and width.

```
pageLayoutObj.PageSize.Orientation = "landscape";
pageLayoutObj.PageSize.Height = "8.5in";
pageLayoutObj.PageSize.Width = "11in";
```

Specify the page margins.

```
pageLayoutObj.PageMargins.Top = "0.5in";
pageLayoutObj.PageMargins.Bottom = "0.5in";
pageLayoutObj.PageMargins.Left = "0.5in";
pageLayoutObj.PageMargins.Right = "0.5in";
```

```
pageLayoutObj.PageMargins.Header = "0.3in";
pageLayoutObj.PageMargins.Footer = "0.3in";
```

Add the page layout object to the report.

```
add(rpt, pageLayoutObj);
```

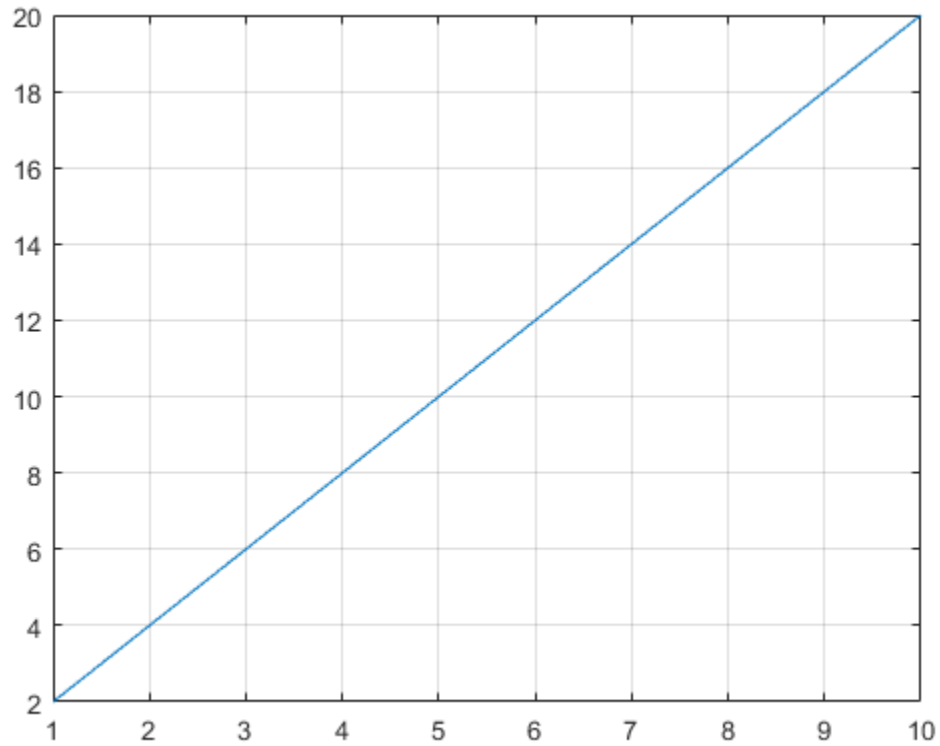
## Create and Add Figures

Create a figure with a linear plot.

```
f = figure();
plot(1:1:10, 2:2:20);
grid on;
```

Call the `centerFigure` local function to add the figure snapshot at the center of the page. Then, delete the figure object.

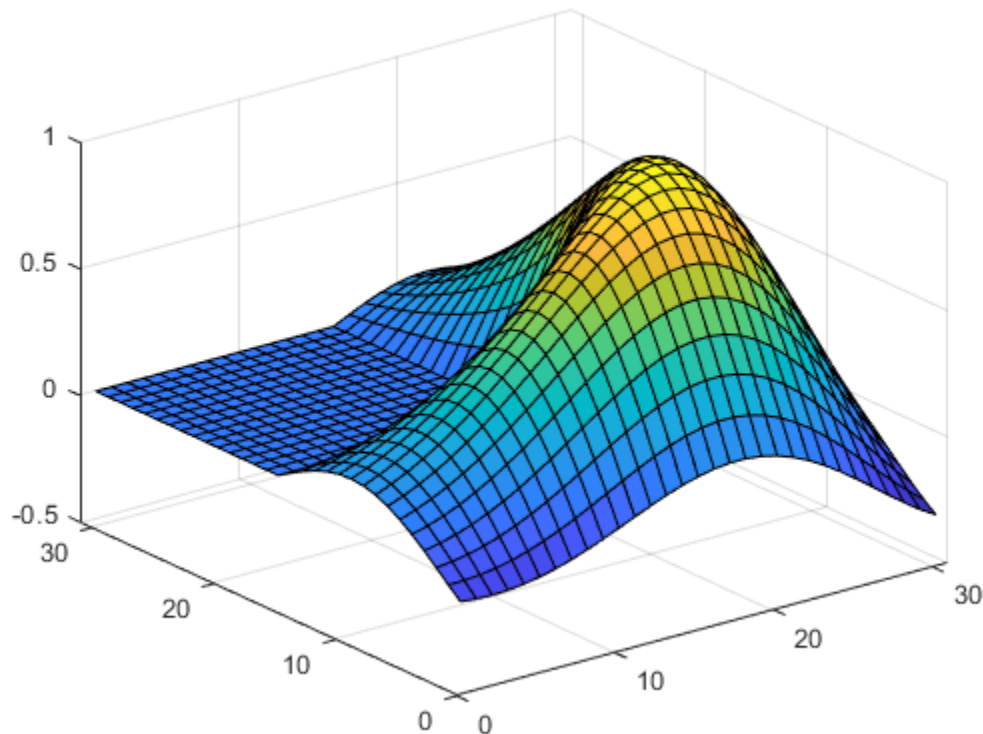
```
centerFigure(f, rpt);
```



```
delete(f);
```

Similarly, add the membrane surface plot snapshot at the center of the page.

```
centerFigure(surf(membrane), rpt);
```



```
delete(gcf);
```

### Generate the Report

Close and view the report.

```
close(rpt);  
rptview(rpt);
```

### The centerFigure Local Function

This function creates a snapshot of the specified figure and adds it at the center of the page in the specified report. The function uses the Report API Figure reporter to take the figure snapshot and an invisible DOM Table to do the layout.

```
function centerFigure(figure,rpt)
```

Import the DOM API, Report API, and report generator utility packages so that you do not have to use long, fully-qualified class names.

```
import mlreportgen.dom.*  
import mlreportgen.report.*  
import mlreportgen.utils.*
```

Get the report's current page layout to determine the current page size and the page margins. The page layout information is used to calculate the page body size in order to size the layout table created in a subsequent step.



```

pageLayout = getReportLayout(rpt);
pageSize = pageLayout.PageSize;
pageMargins = pageLayout.PageMargins;

```

Calculate the page body width. The page body width denotes the page width available for the content and is determined by subtracting the left and right margin size from the page width. For DOCX output, gutter size also needs to be subtracted.

```

bodyWidth = units.toInches(pageSize.Width) - ...
            units.toInches(pageMargins.Left) - ...
            units.toInches(pageMargins.Right);

if strcmpi(rpt.Type,"docx")
    bodyWidth = bodyWidth - ...
                units.toInches(pageMargins.Gutter);
end
bodyWidth = sprintf("%0.2fin",bodyWidth);

```

Calculate the page body height. The page body height denotes the page height available for the content and is determined by subtracting the top and bottom margin size from the page height. For PDF output, the header and footer sizes also need to be subtracted because the body extends from the bottom of the header to the top of the footer.

```

bodyHeight = units.toInches(pageSize.Height) - ...
              units.toInches(pageMargins.Top) - ...
              units.toInches(pageMargins.Bottom);

if strcmpi(rpt.Type,"pdf")
    bodyHeight = bodyHeight - ...
                  units.toInches(pageMargins.Header) - ...
                  units.toInches(pageMargins.Footer);
end
bodyHeight = sprintf("%0.2fin",bodyHeight);

```

Create a Figure object for the specified figure. Then, create an Image object wrapped around the figure snapshot image file. Scale the image to fit the entry of the layout table created in a subsequent step.

```

fig = Figure(figure);
figImg = Image(getSnapshotImage(fig,rpt));
figImg.Style = [figImg.Style {ScaleToFit}];

```

Wrap the image in a paragraph because PDF requires that an image reside in a paragraph. Update the paragraph style to make sure that there is no white space around the image.

```

para = Paragraph(figImg);
para.Style = [para.Style {OuterMargin("0in","0in","0in","0in")}];

```

Add the paragraph that contains the figure snapshot in a 1-by-1 invisible layout table (`lo_table`). A table is considered invisible when the borders are not defined for the table and its table entries.

```

lo_table = Table({para});

```

Span the table to the available page body width.

```

lo_table.Width = bodyWidth;

```

Span the only table entry to the available page body height. Also, specify the vertical and horizontal alignment formats to make sure that the image is centered both vertically and horizontally inside the table entry.

```
lo_table.TableEntriesStyle = [lo_table.TableEntriesStyle ...  
    { ...  
      Height(bodyHeight), ...  
      HAlign("center"), ...  
      VAlign("middle") ...  
    }];
```

Add the layout table to the report.

```
    add(rpt, lo_table);  
end
```

## See Also

[mlreportgen.report.Report](#) | [mlreportgen.dom.PDFPageLayout](#) |  
[mlreportgen.dom.DOCXPageLayout](#) | [getReportLayout](#) | [mlreportgen.utils.units](#) |  
[mlreportgen.report.Figure](#) | [mlreportgen.dom.Table](#)

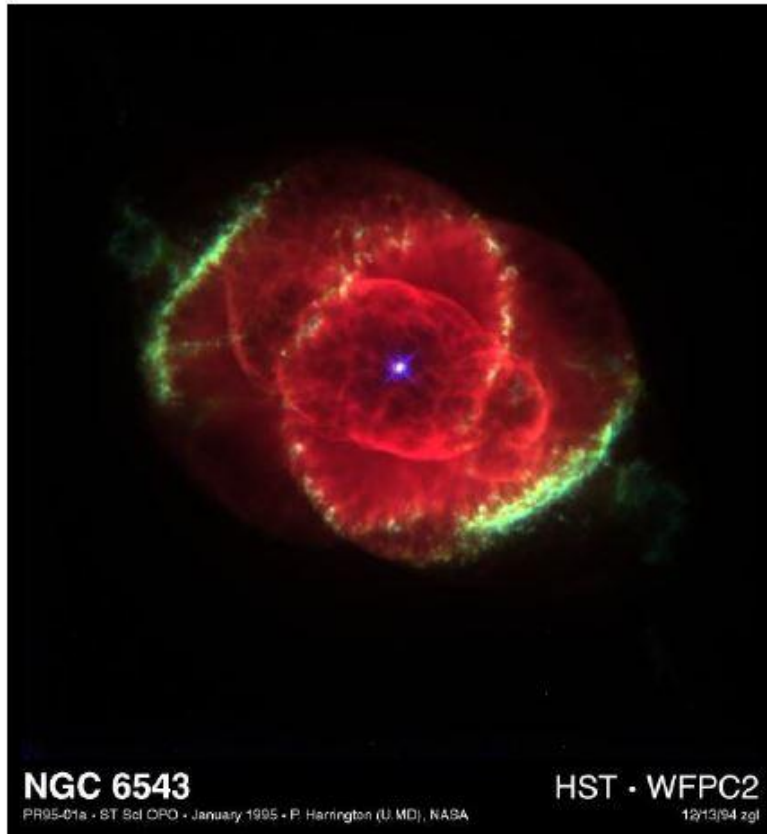
## More About

- “Center Image on a Page” on page 17-121
- “Center Formal Image in a PDF Report” on page 17-126

## Center Image on a Page

This example shows how to center an image on a landscape page for PDF and Microsoft® Word reports.

The example creates a Report API report having landscape layout and uses the `centerImage` local function to add the images at the center of the page.



## Create Report

Import the DOM and Report API packages so that you do not have to use long, fully-qualified class names.

```
import mlreportgen.dom.*
import mlreportgen.report.*
```

Create a PDF report. To create a Microsoft® Word report, change "pdf" to "docx".

```
rpt = Report("myreport", "pdf");
open(rpt);
```

## Update Report Page Layout

Create a page layout object.

```
if strcmpi(rpt.Type, "pdf")
    pageLayoutObj = PDFPageLayout;
else
    pageLayoutObj = DOCXPageLayout;
end
```

Specify the page orientation, height, and width.

```
pageLayoutObj.PageSize.Orientation = "landscape";
pageLayoutObj.PageSize.Height = "8.5in";
pageLayoutObj.PageSize.Width = "11in";
```

Specify the page margins.

```
pageLayoutObj.PageMargins.Top = "0.5in";
pageLayoutObj.PageMargins.Bottom = "0.5in";
pageLayoutObj.PageMargins.Left = "0.5in";
pageLayoutObj.PageMargins.Right = "0.5in";

pageLayoutObj.PageMargins.Header = "0.3in";
pageLayoutObj.PageMargins.Footer = "0.3in";
```

Add the page layout object to the report.

```
add(rpt, pageLayoutObj);
```

## Add Images

Call the centerImage local function to add the images at the center of the page.

```
centerImage(which("ngc6543a.jpg"), rpt);
centerImage(which("peppers.png"), rpt);
```

## Generate the Report

Close and view the report.

```
close(rpt);
rptview(rpt);
```

### The centerImage Local Function

This function adds the specified image at the center of the page in the specified report. The function uses the DOM Image to wrap the image file and an invisible DOM Table to do the layout.

```
function centerImage(imageFile,rpt)
```

Import the DOM API and report generator utility packages so that you do not have to use long, fully-qualified class names.

```
import mlreportgen.dom.*
import mlreportgen.utils.*
```

Get the report's current page layout to determine the current page size and the page margins. The page layout information is used to calculate the page body size in order to size the layout table created in a subsequent step.

```
pageLayout = getReportLayout(rpt);
pageSize = pageLayout.PageSize;
pageMargins = pageLayout.PageMargins;
```

Calculate the page body width. The page body width denotes the page width available for the content and is determined by subtracting the left and right margin size from the page width. For DOCX output, gutter size also needs to be subtracted.

```
bodyWidth = units.toInches(pageSize.Width) - ...
    units.toInches(pageMargins.Left) - ...
    units.toInches(pageMargins.Right);

if strcmpi(rpt.Type,"docx")
    bodyWidth = bodyWidth - ...
        units.toInches(pageMargins.Gutter);
end
bodyWidth = sprintf("%0.2fin",bodyWidth);
```

Calculate the page body height. The page body height denotes the page height available for the content and is determined by subtracting the top and bottom margin size from the page height. For PDF output, the header and footer sizes also need to be subtracted because the body extends from the bottom of the header to the top of the footer.

```
bodyHeight = units.toInches(pageSize.Height) - ...
    units.toInches(pageMargins.Top) - ...
    units.toInches(pageMargins.Bottom);

if strcmpi(rpt.Type,"pdf")
    bodyHeight = bodyHeight - ...
        units.toInches(pageMargins.Header) - ...
        units.toInches(pageMargins.Footer);
end
bodyHeight = sprintf("%0.2fin",bodyHeight);
```

Create an Image object wrapped around the image file. Scale the image to fit the entry of the layout table created in a subsequent step.

```
image = Image(imageFile);
image.Style = [image.Style {ScaleToFit}];
```

Wrap the image in a paragraph because PDF requires that an image reside in a paragraph. Update the paragraph style to make sure that there is no white space around the image.

```
para = Paragraph(image);
para.Style = [para.Style {OuterMargin("0in","0in","0in","0in")}];
```

Add the paragraph that contains the image in a 1-by-1 invisible layout table (`lo_table`). A table is considered invisible when the borders are not defined for the table and its table entries.

```
lo_table = Table({para});
```

Span the table to the available page body width.

```
lo_table.Width = bodyWidth;
```

Span the only table entry to the available page body height. Also, specify the vertical and horizontal alignment formats to make sure that the image is centered both vertically and horizontally inside the table entry.

```
lo_table.TableEntriesStyle = [lo_table.TableEntriesStyle ...
    { ...
      Height(bodyHeight), ...
      HAlign("center"), ...
      VAlign("middle") ...
    }];
```

Add the layout table to the report.

```
add(rpt,lo_table);
end
```

## See Also

`mlreportgen.report.Report` | `mlreportgen.dom.PDFPageLayout` |  
`mlreportgen.dom.DOCXPageLayout` | `getReportLayout` | `mlreportgen.utils.units` |  
`mlreportgen.dom.Image` | `mlreportgen.dom.Table`

## More About

- “Center Figure Snapshot on a Page” on page 17-114
- “Center Formal Image in a PDF Report” on page 17-126

## Center Formal Image in a PDF Report

This example shows how to center a formal image, which is an image with a caption, on a landscape page for PDF reports.

The example creates a Report API PDF report that has landscape layout and uses the `centerFormalImage` local function to add the formal images at the center of the page.



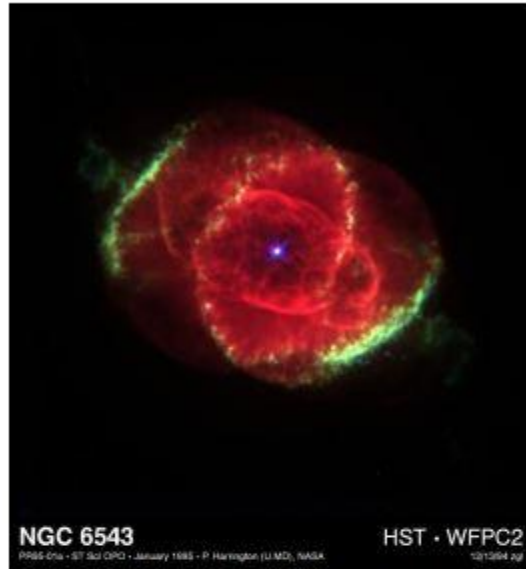


Figure 1. Cat's Eye Nebula or NGC 6543



Figure 2. Peppers

## Create Report

Import the DOM and Report API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
import mlreportgen.report.*
```

Create a PDF report.

```
rpt = Report("myreport", "pdf");
open(rpt);
```

## Update the Report Page Layout

Create a page layout object.

```
pageLayoutObj = PDFPageLayout;
```

Specify the page orientation, height, and width.

```
pageLayoutObj.PageSize.Orientation = "landscape";
pageLayoutObj.PageSize.Height = "8.5in";
pageLayoutObj.PageSize.Width = "11in";
```

Specify the page margins.

```
pageLayoutObj.PageMargins.Top = "0.5in";
pageLayoutObj.PageMargins.Bottom = "0.5in";
pageLayoutObj.PageMargins.Left = "0.5in";
pageLayoutObj.PageMargins.Right = "0.5in";
```

```
pageLayoutObj.PageMargins.Header = "0.3in";
pageLayoutObj.PageMargins.Footer = "0.3in";
```

Add the page layout object to the report.

```
add(rpt, pageLayoutObj);
```

## Create and Add Formal Images

Create formal images and call the `centerFormalImage` local function to add these formal images at the center of the page.

```
formalImg1 = FormalImage( ...
    "Image", which("ngc6543a.jpg"), ...
    "Caption", "Cat's Eye Nebula or NGC 6543");
centerFormalImage(formalImg1, rpt);
```

```
formalImg2 = FormalImage( ...
    "Image", which("peppers.png"), ...
    "Caption", "Peppers");
centerFormalImage(formalImg2, rpt);
```

## Generate the Report

Close and view the report.

```
close(rpt);
rptview(rpt);
```

### The centerFormalImage Local Function

This function adds the specified formal image at the center of the page in the specified PDF report. The function uses an invisible DOM Table to do the layout.

```
function centerFormalImage(formalImage,rpt)
```

Import the DOM API, Report API, and report generator utility packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*
import mlreportgen.report.*
import mlreportgen.utils.*
```

To determine the current page size and the page margin, get the current page layout of the report. The page layout information is used to size the table that is created in a subsequent step.

```
pageLayout = getReportLayout(rpt);
pageSize = pageLayout.PageSize;
pageMargins = pageLayout.PageMargins;
```

Calculate the page body width. The page body width denotes the page width that is available for the content. For a PDF report, the page body width is determined by subtracting the left and right margin size from the page width.

```
bodyWidth = units.toInches(pageSize.Width) - ...
            units.toInches(pageMargins.Left) - ...
            units.toInches(pageMargins.Right);
bodyWidth = sprintf("%0.2fin",bodyWidth);
```

Calculate the page body height. The page body height denotes the page height that is available for the content. For a PDF report, the page body height is determined by subtracting the top margin, bottom margin, header, and footer size from the page height.

```
bodyHeight = units.toInches(pageSize.Height) - ...
             units.toInches(pageMargins.Top) - ...
             units.toInches(pageMargins.Bottom) - ...
             units.toInches(pageMargins.Header) - ...
             units.toInches(pageMargins.Footer);
bodyHeight = sprintf("%0.2fin",bodyHeight);
```

Use the `getImageReporter` method of the `FormalImage` reporter to get the image reporter and the `getCaptionReporter` method to get the caption reporter.

```
imageReporter = getImageReporter(formalImage,rpt);
captionReporter = getCaptionReporter(formalImage);
```

Use the `getImpl` methods of the image and caption reporters to get the corresponding DOM implementations.

```
imageImpl = getImpl(imageReporter,rpt);
captionImpl = getImpl(captionReporter,rpt);
```

The DOM implementations contain a DOM Paragraph that contains the image and caption content. Update the style of the paragraphs to make sure that there is no white space around the paragraphs and that they are centered in the table entry that is created in a subsequent step.

```
paraStyle = { ...
    OuterMargin("0in","0in","0in","0in"), ...
    HAlign("center") ...
};

imagePara = clone(imageImpl.Children(1));
imagePara.Style = [imagePara.Style, paraStyle];

captionPara = clone(captionImpl.Children(1));
captionPara.Style = [captionPara.Style, paraStyle];
```

Create a 1-by-1 invisible layout table (`lo_table`). A table is considered invisible when the borders are not defined for the table and its table entries.

```
lo_table = Table(1);
row = append(lo_table,TableRow);
entry = append(row,TableEntry);
```

Add the paragraphs that contain the image and caption content to the only table entry in the invisible layout table.

```
append(entry,imagePara);
append(entry,captionPara);
```

Span the table to the available page body width.

```
lo_table.Width = bodyWidth;
```

Span the only table entry to the available page body height. Specify the vertical and horizontal alignment formats to make sure that the image and caption are centered both vertically and horizontally inside the table entry.

```
lo_table.TableEntriesStyle = [lo_table.TableEntriesStyle ...
    { ...
    Height(bodyHeight), ...
    HAlign("center"), ...
    VAlign("middle") ...
    }];
```

Add the layout table to the report.

```
add(rpt,lo_table);
end
```

## See Also

[mlreportgen.report.Report](#) | [mlreportgen.dom.PDFPageLayout](#) | [getReportLayout](#) | [mlreportgen.utils.units](#) | [mlreportgen.report.FormalImage](#) | [mlreportgen.dom.Table](#)

## More About

- “Center Image on a Page” on page 17-121
- “Center Figure Snapshot on a Page” on page 17-114

## Define Styles Programmatically

This example shows how to define a set of styles programmatically. A style is a named set of formats. Styles allow you to define the appearance of report elements by specifying style names instead of defining all of the individual formats. Using styles can simplify the creation of report programs. The DOM API allows you to use styles that you define in a report template. Alternatively, as this example shows, you can define styles in a report program. Consider defining styles in a report program if you need or prefer to create a report program that does not require a template.

In this example, you generate a report that contains a table that summarizes the results of a sequence of tests. The example creates the test data. In a real-world application, you can implement the report program as a function that accepts the test data as one of its inputs. Here is the report generated by this example:

### Load Test Information

#### *Quality of Service Report Summary*

| Name               | Description                                     | Status | Status Detail           |
|--------------------|-------------------------------------------------|--------|-------------------------|
| 100% Success       | Percent of total loss(failing hits) must be==0% | Fail   | Loss=22.33%             |
| No Fail            | Fail Count must be ==0                          | Fail   | Fail Count=69           |
| Fast Hit Rate      | Final Throughput must be>=10Hits per second     | Pass   | Hits per Second=27.978  |
| Low Execution Time | Avg. Exe Time(ms) must be <750.0                | Pass   | Avg Exe Time(ms)=244.33 |

### Create Test Data

Create the test data as an array of structures.

```
test(1).Name = "100% Success";
test(1).Description = "Percent of total loss(failing hits) must be==0%";
test(1).Status = "Fail";
test(1).StatusDetail = "Loss=22.33%";

test(2).Name = "No Fail";
test(2).Description = "Fail Count must be ==0";
test(2).Status = "Fail";
test(2).StatusDetail = "Fail Count=69";

test(3).Name = "Fast Hit Rate";
test(3).Description = "Final Throughput must be>=10Hits per second";
test(3).Status = "Pass";
test(3).StatusDetail = "Hits per Second=27.978";
```

```
test(4).Name = "Low Execution Time";
test(4).Status = "Pass";
test(4).Description = "Avg. Exe Time(ms) must be <750.0";
test(4).StatusDetail = "Avg Exe Time(ms)=244.33";
```

### Import DOM API Package

Import the DOM API package so that you do not have to use fully qualified class names.

```
import mlreportgen.dom.*;
```

### Create a Document

To create a Microsoft® Word document, change the output type from pdf to docx. To create an HTML document, change pdf to html or html-file.

```
d = Document("report", "pdf");
```

### Create a Container for the Styles

Create a MATLAB map object to serve as a style sheet, that is, an object that maps style names to the set of formats that the styles contain.

```
styles = containers.Map;
```

### Define the Styles

This example uses a style for a top-level section heading style and a style for a subsection heading. First, create a style that defines the appearance common to both headings.

```
styles("baseHeadingPara") = {Color("darkblue"), FontFamily("Arial")};
```

Define a style for a paragraph that serves as a heading of a top-level report section. This style specifies an outline level of 1 so that the heading appears as a top-level entry in the report table of contents.

```
styles("heading1Para") = [styles("baseHeadingPara"), {OutlineLevel(1), Bold, ...
    FontSize("16pt")}];
```

Define a second level heading.

```
styles("heading2Para") = [styles("baseHeadingPara"), {OutlineLevel(2), ...
    OuterMargin("0in", "0in", "12pt", "5pt"), Italic, FontSize("14pt")}];
```

Define the styles for a table, table header, and table entries.

```
styles("testSummaryTable") = {Border("solid"), RowSep("solid"), ColSep("solid"), Width("7in")};
styles("testSummaryTableHeader") = {Bold};
styles("testSummaryTableEntry") = {InnerMargin("10pt")};
```

Define the styles for the test status.

```
styles("passText") = {Color("green")};
styles("failText") = {Color("red")};
```

### Create the Report Content

Create the report title.

```
heading1Para = Paragraph("Load Test Information");
heading1Para.Style = styles("heading1Para");
append(d,heading1Para);
```

Create the test summary table description title.

```
heading2Para = Paragraph("Quality of Service Report Summary");
heading2Para.Style = styles("heading2Para");
append(d,heading2Para);
```

Create the test summary table header content. Define labels to display in the table header.

```
testSummaryTableHeader = ["Name", "Description", "Status", "Status Detail"];
```

Create a cell array to hold the content of the test summary table. The cell array is used later to create the table.

```
nTests = numel(test);
testSummaryTableBody = cell(nTests,4);
```

Store each item of the test data in a cell of the cell array.

```
for t = 1:nTests
    testSummaryTableBody{t,1} = test(t).Name;
    testSummaryTableBody{t,2} = test(t).Description;
    result = test(t).Status;
    status = Paragraph(result);
    if result == "Pass"
        status.Style = styles("passText");
    else
        status.Style = styles("failText");
    end
    testSummaryTableBody{t,3} = status;
    testSummaryTableBody{t,4} = test(t).StatusDetail;
end
```

Create the test summary table.

```
testSummaryTable = FormalTable(testSummaryTableHeader, testSummaryTableBody);
testSummaryTable.Style = styles("testSummaryTable");
testSummaryTable.Header.Style = styles("testSummaryTableHeader");
testSummaryTable.TableEntriesStyle = styles("testSummaryTableEntry");
```

Append the test summary table to the document.

```
append(d, testSummaryTable);
```

### Close and View the Document

```
close(d);
rptview(d);
```

### See Also

[mlreportgen.dom.Color](#) | [mlreportgen.dom.Document](#) | [mlreportgen.dom.FontFamily](#) | [mlreportgen.dom.OutlineLevel](#) | [mlreportgen.dom.Paragraph](#) | [mlreportgen.dom.FormalTable](#) | [mlreportgen.dom.FontSize](#) | [mlreportgen.dom.RowSep](#) | [mlreportgen.dom.InnerMargin](#) | [mlreportgen.dom.Border](#) | [mlreportgen.dom.OuterMargin](#)

## **More About**

- “Report Formatting Approaches” on page 13-17



## Customize the Page Size and Margins of a Word Report Using Templates

This example shows how to use Report API reporter templates to customize the page size and the sizes of the page margins, header, footer, and gutter in a Microsoft® Word report. Alternatively, you can use a programmatic approach. Use the template-based approach if you plan to make other modifications to the templates. Otherwise, use the programmatic approach. For an example that uses the programmatic approach, see “Customize the Page Size and Margins of a Report Programmatically” on page 17-154.

By default, the Report API generates 8.5-by-11-inch portrait pages with one-inch margins. Headers and footers are each one-half inch wide and the gutter is zero pixels. You may need to change these dimensions in your report. For example, if you are generating a report for a European locale, you may need to generate A4 pages. This example shows how to generate A4 pages. Use the same procedure to generate other page dimensions.

To change the page dimensions of a Report API report, you must change the dimensions specified by the templates of the `mlreportgen.report.TitlePage`, `mlreportgen.report.TableOfContents`, and `mlreportgen.report.Chapter` reporters. This example shows how to modify and use the Word template for each reporter.

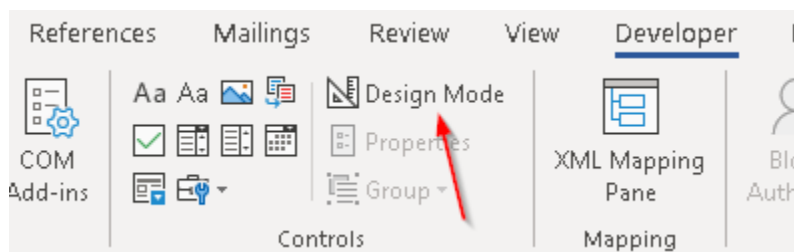
### Customize the Title Page Word Template

Create a copy of the `mlreportgen.report.TitlePage` reporter default Word template.

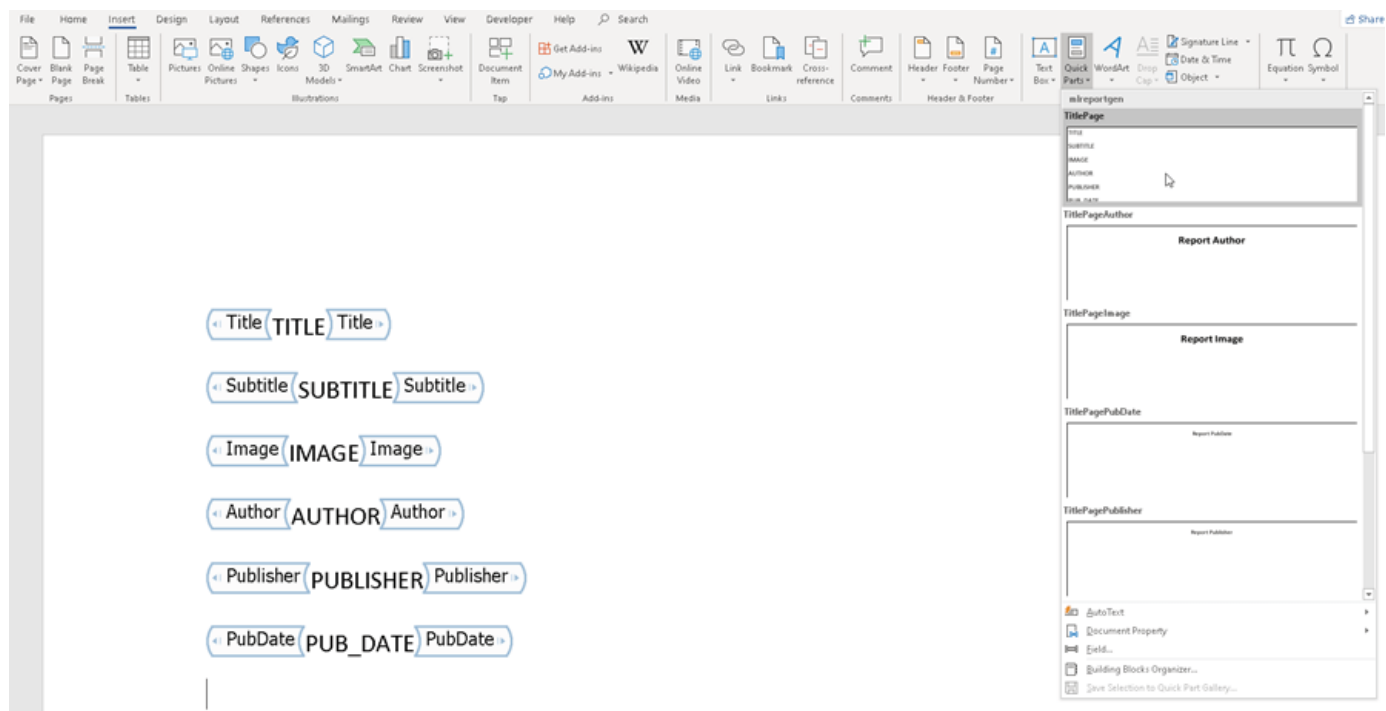
```
mlreportgen.report.TitlePage.createTemplate("titlePageTemplate.dotx", "docx");
```

In MATLAB®, in the **Current Folder** pane, right-click the template file and click **Open Outside MATLAB**.

On the **Developer** tab, enable **Design Mode**. If the **Developer** tab is not available, click **File > Options**, and then click **Customize Ribbon**. Under **Main Tabs**, click the **Developer** check box.



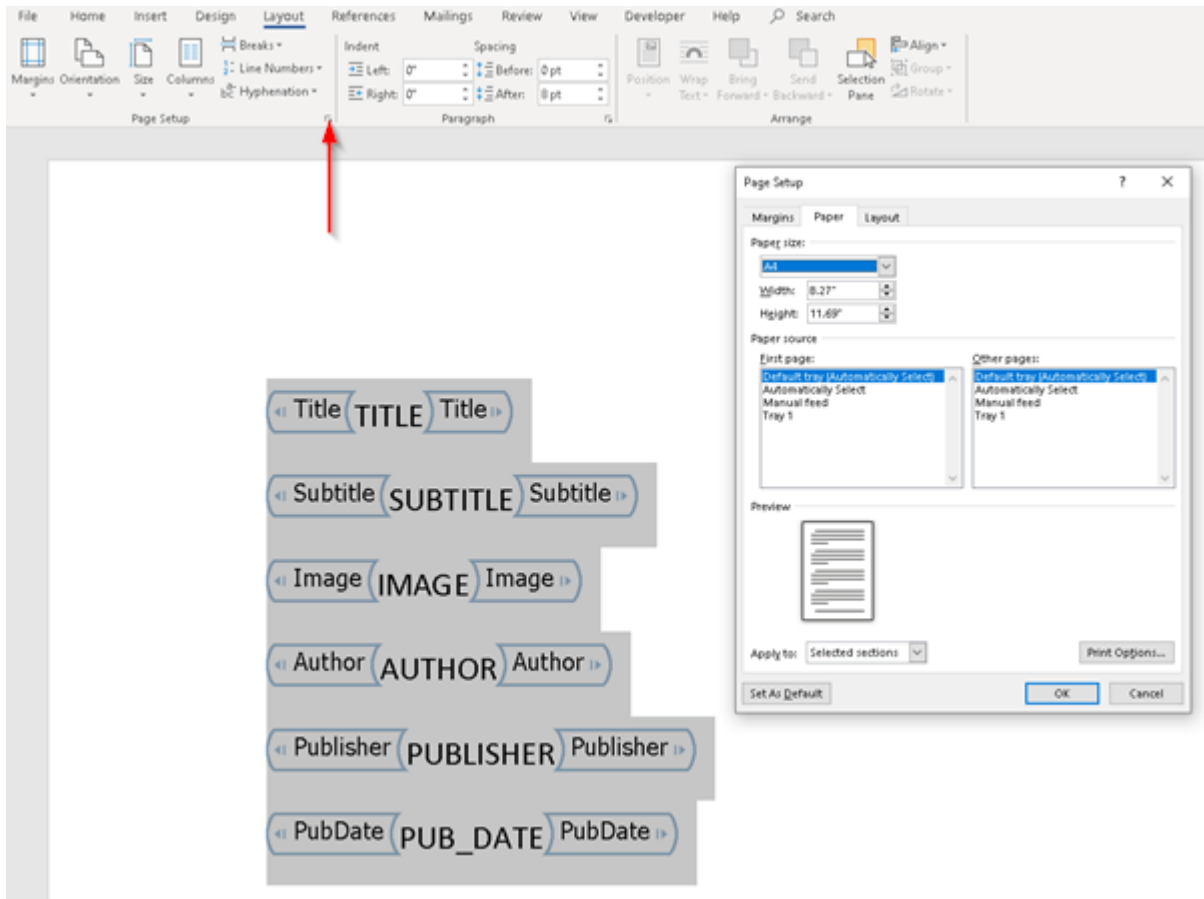
On the **Insert** tab, in the **Text** group, click **Quick Parts**. Select the `TitlePage` document part template.



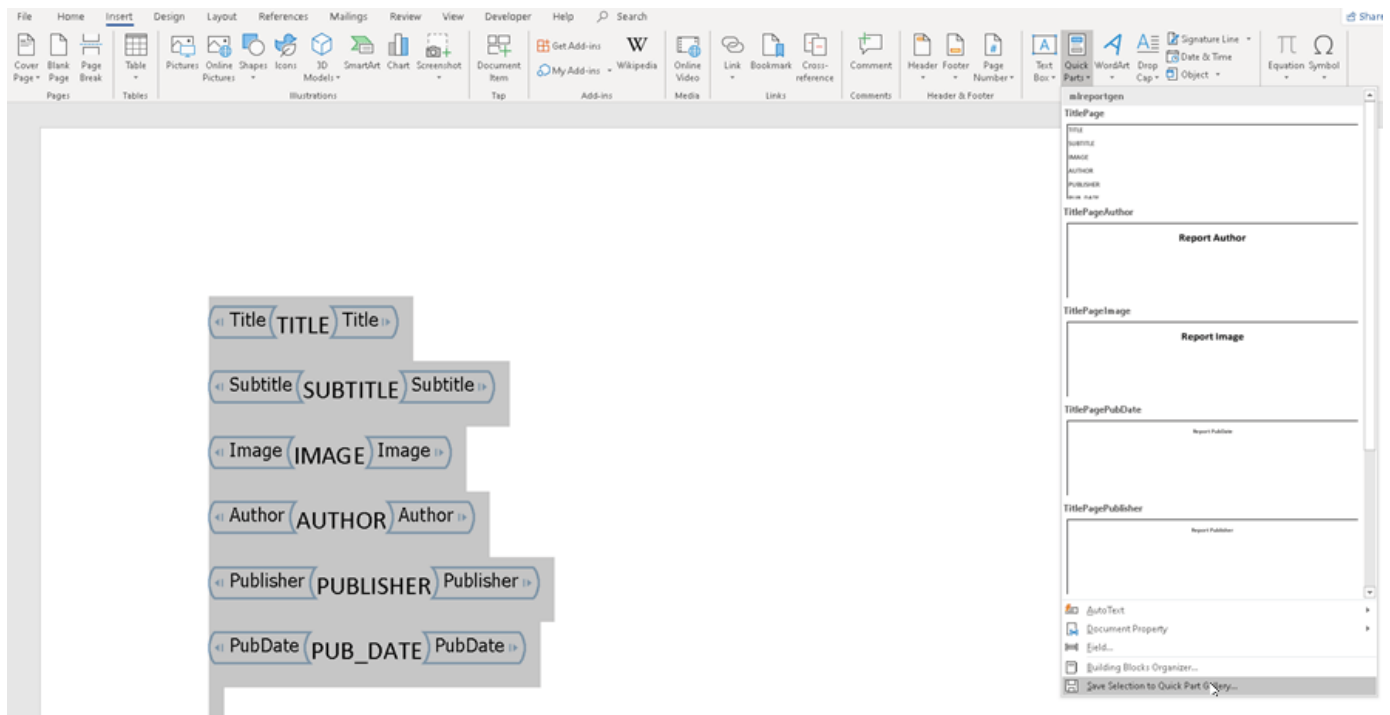
Select the content of the document part template. On the **Layout** tab, in the **Page Setup** group, click the **Page Setup** dialog box launcher. Modify the paper size and margins to A4 values:

- Top, bottom, left, right margins of 0.98 inches.
- Header and footer heights of 0.5 inches.
- Gutter size (space for binding pages) of 0.
- 8.27-inch by 11.69-inch page size in portrait orientation.

Click **OK** and close the dialog box.



On the **Insert** tab, click **Quick Parts** and select **Save Selection to the Quick Part Gallery**.



In the Create New Building Block dialog box, enter the name of the document part template you modified, that is, `TitlePage` and select the `m1reportgen` category. Respond to the prompt to overwrite the previous version.

Select the content of the document part template, and save and close the template.

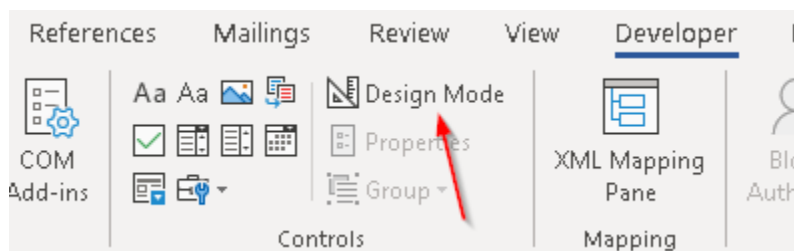
### Customize the Table of Contents Word Template

Create a copy of the `m1reportgen.report.TableOfContents` reporter default Word template.

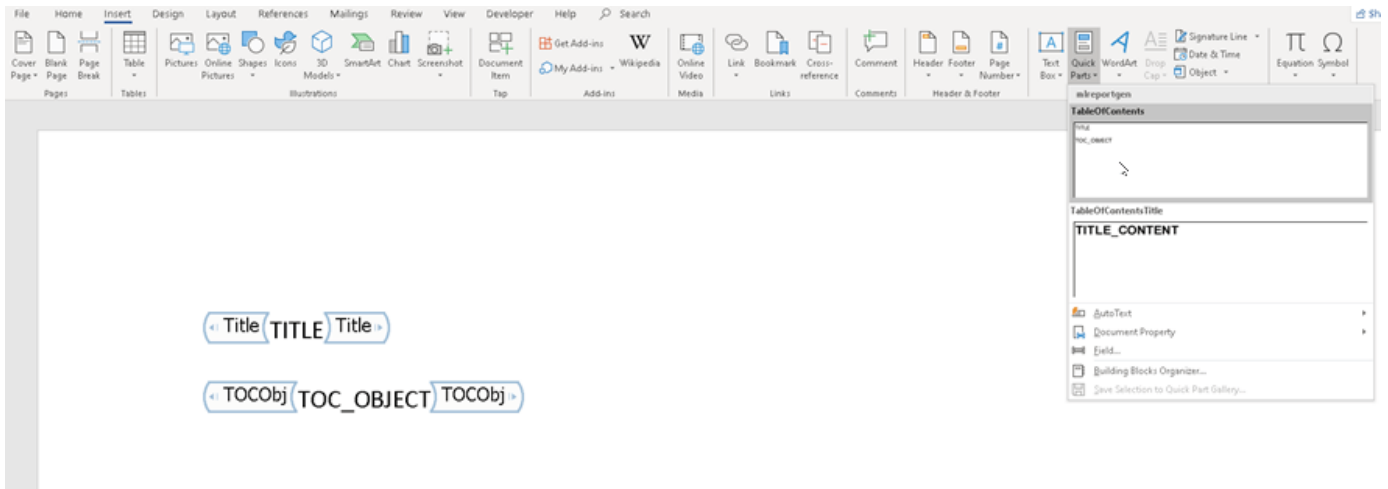
```
m1reportgen.report.TableOfContents.createTemplate("tableOfContentsTemplate.dotx", "docx");
```

In MATLAB, in the **Current Folder** pane, right-click the template file and click **Open Outside MATLAB**.

On the **Developer** tab, enable **Design Mode**. If the **Developer** tab is not available, click **File > Options**, and then click **Customize Ribbon**. Under **Main Tabs**, click the **Developer** check box.



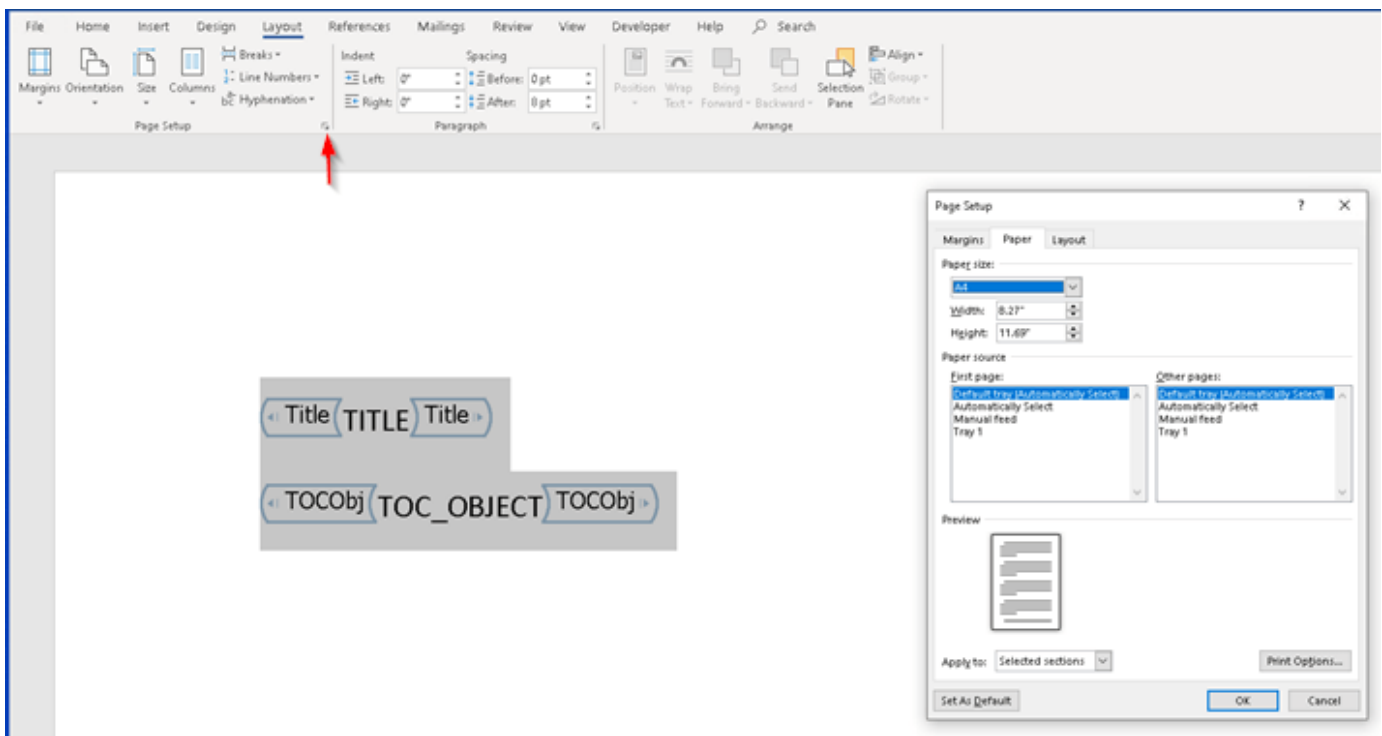
On the **Insert** tab, click **Quick Parts**. Select the `TableOfContents` document part template.



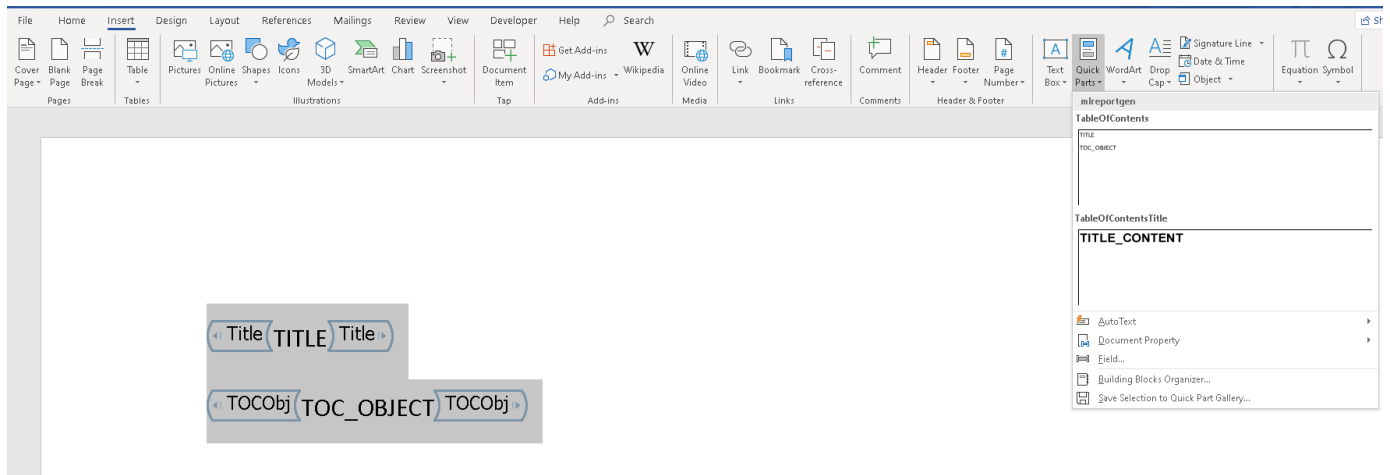
Select the content of the document part template. On the **Layout** tab, in the **Page Setup** group, click the **Page Setup** dialog box launcher. Modify the paper size and margins to the A4 values:

- Top, bottom, left, right margins of 0.98 inches.
- Header and footer heights of 0.5 inches.
- Gutter size (space for binding pages) of 0.
- 8.27-inch by 11.69-inch page size in portrait orientation.

Click **OK** and close the dialog box.



On the **Insert** tab, click **Quick Parts** and select **Save Selection to the Quick Part Gallery**.



In the Create New Building Block dialog box, enter the name of the document part template you modified, that is, TableOfContents and select the m1reportgen category. Respond to the prompt to overwrite the previous version.

Delete the content of the document part template, and save and close the template.

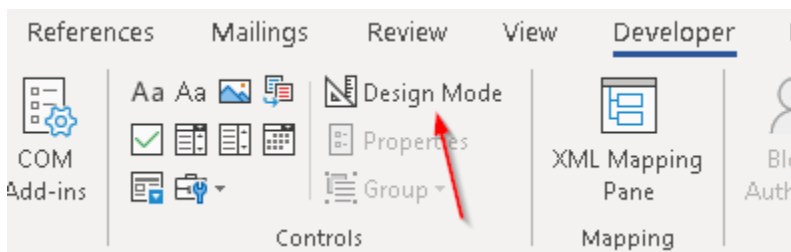
### Customize the Chapter Word Template

Create a copy of the m1reportgen.report.Chapter.reporter default Word template.

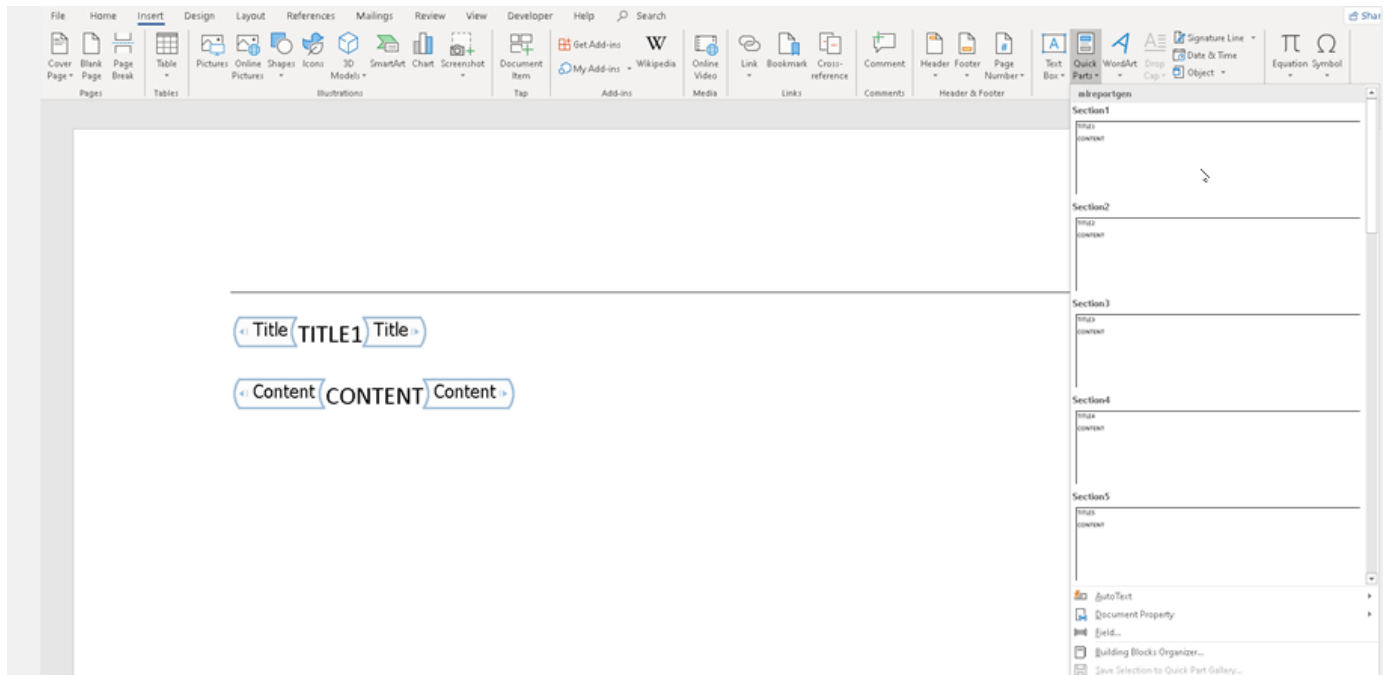
```
m1reportgen.report.Chapter.createTemplate("chapterTemplate.dotx", "docx");
```

In MATLAB®, in the **Current Folder** pane, right-click the template file and click **Open Outside MATLAB**.

On the **Developer** tab, enable **Design Mode**. If the **Developer** tab is not available, click **File > Options**, and then click **Customize Ribbon**. Under **Main Tabs**, click the **Developer** check box.



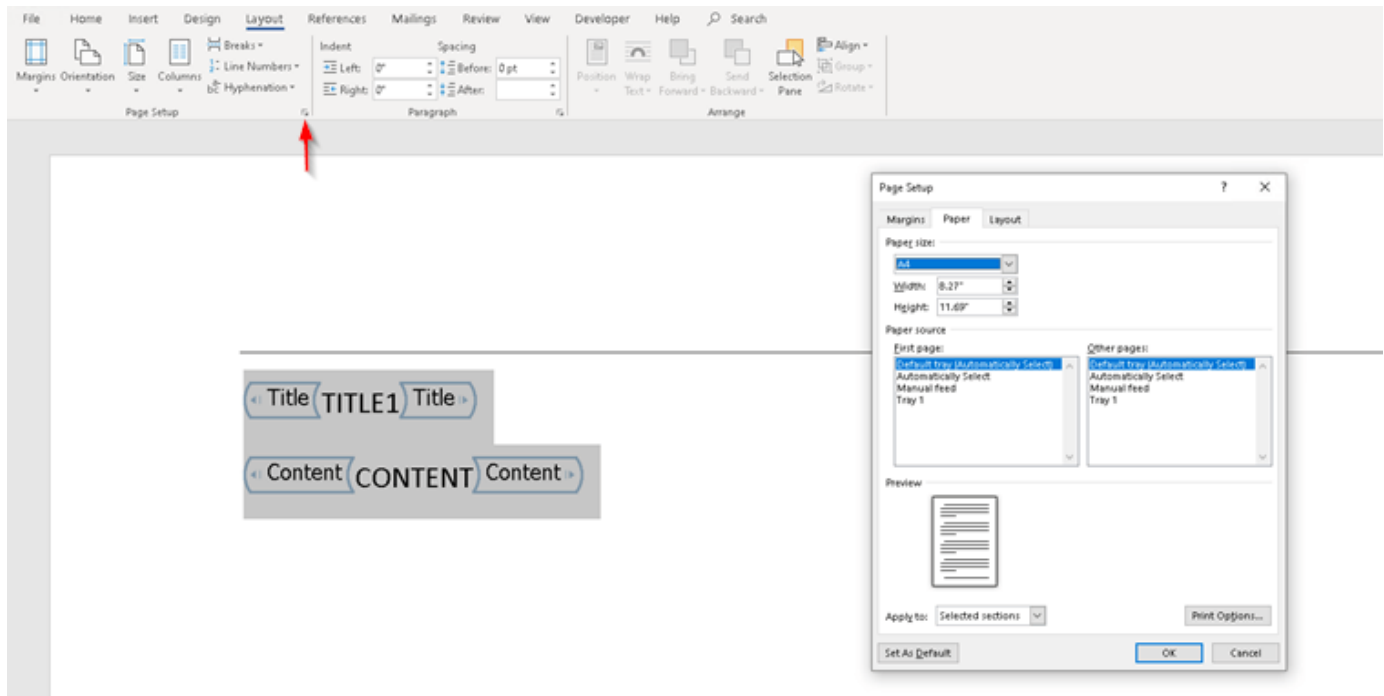
On the **Insert** tab, click **Quick Parts**. Select the Section1 document part template.



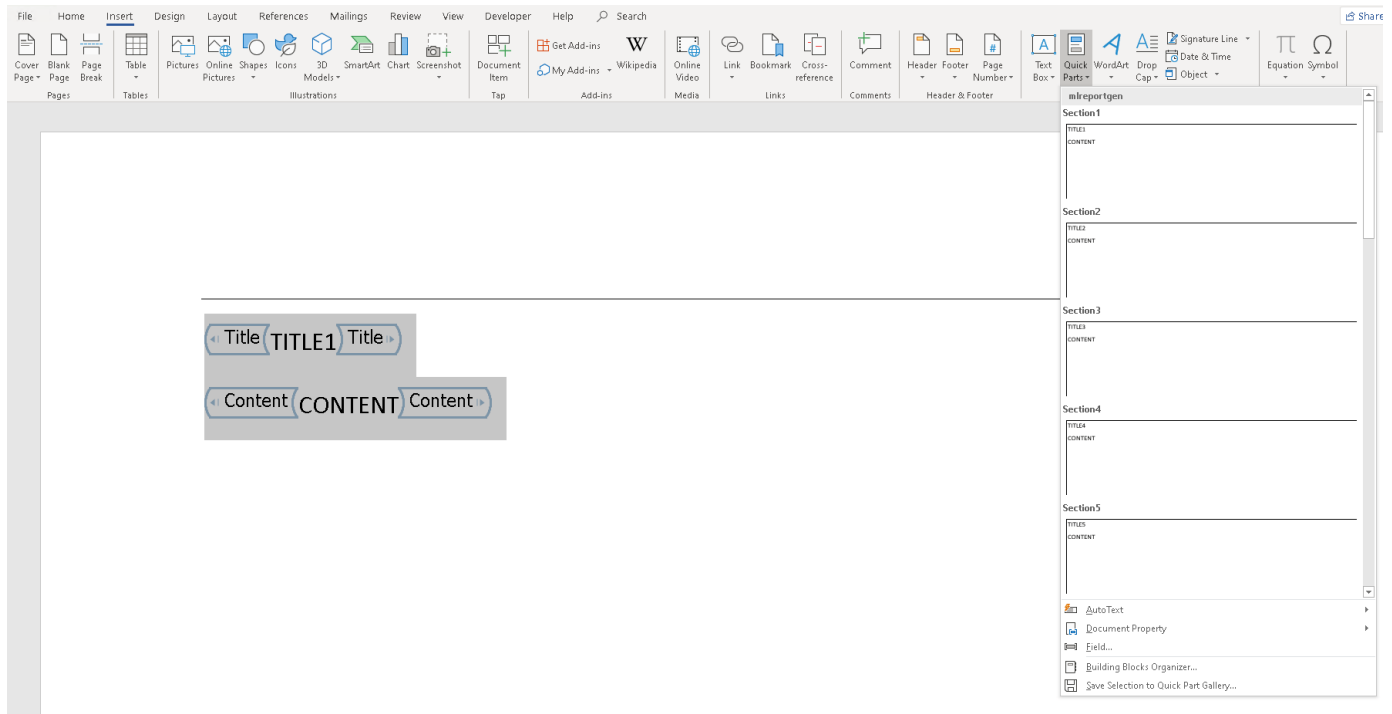
Select the content of the document part template. On the **Layout** tab, in the **Page Setup** group, click the **Page Setup** dialog box launcher. Modify the paper size and margins to the A4 values:

- Top, bottom, left, right margins of 0.98 inches.
- Header and footer heights of 0.5 inches.
- Gutter size (space for binding pages) of 0.
- 8.27-inch by 11.69-inch page size in portrait orientation.

Click **OK** and close the dialog box.



On the **Insert** tab, click **Quick Parts** and select **Save Selection to the Quick Part Gallery**.



In the Create New Building Block dialog box, enter the name of the document part template you modified, that is, **Section1** and select the **mReportgen** category. Respond to the prompt to overwrite the previous version.



Delete the content of the document part template, and save and close the template.

### Use Customized Templates in a Report

Use the customized template to create a Word report on magic squares, that is, matrices whose columns, rows, and diagonals each add up to the same number.

### Import DOM and Report API Packages

Import the DOM and Report API packages to avoid having to use fully qualified class names.

```
import mlreportgen.report.*;
import mlreportgen.dom.*;
```

Create a report container.

```
rpt = Report("report", "docx");
```

### Add a Title Page

Create a title page and specify the title, subtitle, and author.

```
title = TitlePage("Title", "Magic Squares");
title.Subtitle = "Columns, Rows, Diagonals: All Equal Sums";
title.Author = "Albrecht Durer";
```

Use the custom title page template to generate the title page.

```
title.TemplateSrc= fullfile(pwd, "titlePageTemplate.dotx");
```

Add the title page to the report.

```
append(rpt, title);
```

### Add a Table of Contents

```
toc = TableOfContents;
```

Use the custom table of contents template to generate the table of contents.

```
toc.TemplateSrc = fullfile(pwd, "tableOfContentsTemplate.dotx");
```

Add the table of contents to the report.

```
append(rpt, toc);
```

### Add a Chapter

Create a chapter and specify the title.

```
chapter = Chapter("Introduction");
```

Use the custom chapter template to generate the chapter.

```
chapter.TemplateSrc = fullfile(pwd, "chapterTemplate.dotx");
```

Create a section and add a paragraph to it.

```
sec1 = Section("What is a Magic Square?");
para = Paragraph(['A magic square is an N-by-N matrix '...]);
```

```
'constructed from the integers 1 through N^2 '...  
'with equal row, column, and diagonal sums.'];  
append(sec1,para)
```

Add the section to the chapter.

```
append(chapter, sec1)
```

Create another section and add a paragraph to it.

```
sec2 = Section("Albrecht Durer and the Magic Square");  
para = Paragraph([ ...  
'The German artist Albrecht Durer (1471-1528) created '...  
'many woodcuts and prints with religious and '...  
'scientific symbolism. One of his most famous works, '...  
'Melancholia I, explores the depressed state of mind '...  
'which opposes inspiration and expression. '...  
'Renaissance astrologers believed that the Jupiter '...  
'magic square (shown in the upper right portion of '...  
'the image) could aid in the cure of melancholy. The '...  
'engraving's date (1514) can be found in the '...  
'lower row of numbers in the square.']);  
append(sec2,para);
```

Add the section to the chapter.

```
append(chapter, sec2);
```

Add the chapter to the report.

```
append(rpt, chapter);
```

### **Close and View the Document**

```
close(rpt);  
rptview(rpt);
```

### **See Also**

`mlreportgen.report.TitlePage` | `mlreportgen.report.TableOfContents` |  
`mlreportgen.report.Chapter`

### **More About**

- “Customize the Page Size and Margins of a PDF Report Using Templates” on page 17-145
- “Customize the Page Size and Margins of a Report Programmatically” on page 17-154
- “Report Formatting Approaches” on page 13-17

## Customize the Page Size and Margins of a PDF Report Using Templates

This example shows how to use Report API reporter templates to customize the page size and the sizes of the page margins, header, footer, and gutter in a PDF report. Alternatively, you can use a programmatic approach. Use the template-based approach if you plan to make other modifications to the templates. Otherwise, use the programmatic approach. For an example that uses the programmatic approach, see “Customize the Page Size and Margins of a Report Programmatically” on page 17-154.

By default, the Report API generates 8.5-by-11-inch portrait pages with one-inch margins. Headers and footers are each one-half inch wide and the gutter is zero pixels. You may need to change these dimensions in your report. For example, if you are generating a report for a European locale, you may need to generate A4 pages. This example shows how to generate A4 pages. Use the same procedure to generate other page dimensions.

To change the page dimensions of a Report API report, you must change the dimensions specified by the templates of the `mlreportgen.report.TitlePage`, `mlreportgen.report.TableOfContents`, and `mlreportgen.report.Chapter` reporters. This example shows how to modify and use the PDF template for each reporter.

### Customize the Title Page PDF Template

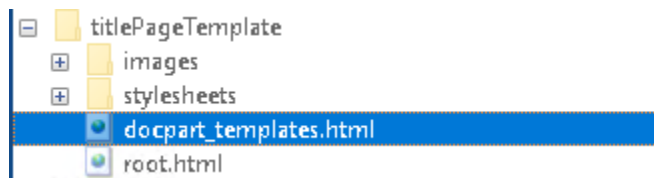
Create a copy of the `mlreportgen.report.TitlePage` reporter default PDF template.

```
mlreportgen.report.TitlePage.createTemplate("titlePageTemplate.pdf", "pdf");
```

Unzip the template.

```
unzipTemplate("titlePageTemplate.pdf");
```

Open the `titlePageTemplate` unzipped folder.



In an HTML or text editor, open the `docpart_templates.html` file.

```
<dptemplate name="TitlePage">
  <layout style="page-margin:0.5in 1in 0.5in 1in 0.5in 0in; page-size: 8.5in 11in portrait">
  </layout>
  <hole id="Title">TITLE</hole>
  <hole id="Subtitle">SUBTITLE</hole>
  <hole id="Image">IMAGE</hole>
  <hole id="Author">AUTHOR</hole>
  <hole id="Publisher">PUBLISHER</hole>
  <hole id="PubDate">PUBLICATION_DATE</hole>
</dptemplate>
```

To modify the page size and margins to A4 values, modify the `page-margin` and `page-size` properties of the `style` attribute of the `layout` element.

Set the `page-margin` property value to:

```
0.98in 0.98in 0.98in 0.98in 0.5in 0.5in 0in
```

This value specifies the margins in this order:

- 1 Top margin of 0.98 inches.
- 2 Left margin of 0.98 inches
- 3 Bottom margin of 0.98 inches
- 4 Right margin of 0.98 inches
- 5 Header of 0.5 inches
- 6 Footer of 0.5 inches
- 7 Gutter size of 0

Set the `page-size` property to:

```
8.27in 11.69in portrait
```

This value specifies an 8.27-by-11.69 inch page size in portrait orientation.

The template now looks like this:

```

<dptemplate name="TitlePage">
  <layout style="page-margin:0.98in 0.98in 0.98in 0.98in 0.5in 0.5in 0in; page-size: 8.27in 11.69in portrait">
  </layout>
  <hole id="Title">TITLE</hole>
  <hole id="Subtitle">SUBTITLE</hole>
  <hole id="Image">IMAGE</hole>
  <hole id="Author">AUTHOR</hole>
  <hole id="Publisher">PUBLISHER</hole>
  <hole id="PubDate">PUBLICATION_DATE</hole>
</dptemplate>

```

Save the docpart\_templates.html file.

Zip the template.

```
zipTemplate("titlePageTemplate.pdf");
```

### Customize the Table of Contents PDF Template

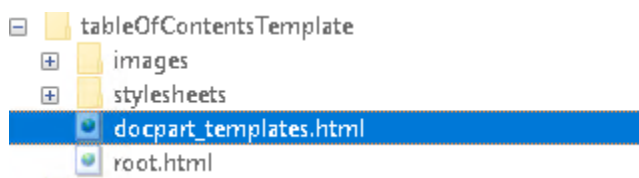
Create a copy of the mlreportgen.report.TableOfContents reporter default PDF template.

```
mlreportgen.report.TableOfContents.createTemplate("tableOfContentsTemplate.pdf", "pdf");
```

Unzip the template.

```
unzipTemplate("tableOfContentsTemplate.pdf");
```

Open the tableOfContentsTemplate unzipped folder.



In an HTML or text editor, open the docpart\_templates.html file.

```

<dptemplate name="TableOfContents">
  <layout style="page-margin:0.5in 1in 0.5in 1in 0.5in 0.5in 0in; page-size:8.5in 11in portrait">
    <pheader type="first" template-name="rgTableOfContentsFirstPageHeader"/>
    <pfooter type="first" template-name="rgTableOfContentsFirstPageFooter"/>
    <pheader type="default" template-name="rgTableOfContentsDefaultPageHeader"/>
    <pfooter type="default" template-name="rgTableOfContentsDefaultPageFooter"/>
    <pheader type="even" template-name="rgTableOfContentsEvenPageHeader"/>
    <pfooter type="even" template-name="rgTableOfContentsEvenPageFooter"/>
    <pnumber format="i" />
  </layout>
  <hole id="Title" default-style-name="TableOfContentsTitle">TITLE</hole>
  <hole id="TOCObj">TOC_OBJECT</hole>
</dptemplate>

```

To modify the page size and margins to A4 values, modify the `page-margin` and `page-size` properties of the `style` attribute of the `layout` element.

Set the `page-margin` property value to:

```
0.98in 0.98in 0.98in 0.98in 0.5in 0.5in 0in
```

This value specifies the margins in this order:

- 1 Top margin of 0.98 inches.
- 2 Left margin of 0.98 inches
- 3 Bottom margin of 0.98 inches
- 4 Right margin of 0.98 inches
- 5 Header of 0.5 inches
- 6 Footer of 0.5 inches
- 7 Gutter size of 0

Set the `page-size` property to:

```
8.27in 11.69in portrait
```

This value specifies an 8.27-by-11.69 inch page size in portrait orientation.

The template now looks like this:

```

<dptemplate name="TableOfContents">
  <layout style="page-margin: 0.98in 0.98in 0.98in 0.5in 0.5in 0in; page-size: 8.27in 11.69in portrait">
    <pheader type="first" template-name="rgTableOfContentsFirstPageHeader"/>
    <pfooter type="first" template-name="rgTableOfContentsFirstPageFooter"/>
    <pheader type="default" template-name="rgTableOfContentsDefaultPageHeader"/>
    <pfooter type="default" template-name="rgTableOfContentsDefaultPageFooter"/>
    <pheader type="even" template-name="rgTableOfContentsEvenPageHeader"/>
    <pfooter type="even" template-name="rgTableOfContentsEvenPageFooter"/>
    <pnumber format="i" />
  </layout>
  <hole id="Title" default-style-name="TableOfContentsTitle">TITLE</hole>
  <hole id="TOCObj">TOC_OBJECT</hole>
</dptemplate>

```

Save the docpart\_templates.html file.

Zip the template.

```
zipTemplate("tableOfContentsTemplate.pdf");
```

### Customize the Chapter PDF Template

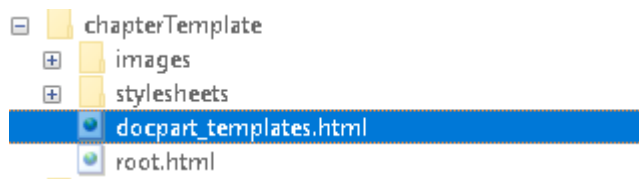
Create a copy of the mlreportgen.report.Chapter reporter default PDF template.

```
mlreportgen.report.Chapter.createTemplate("chapterTemplate.pdf", "pdf");
```

Unzip the template.

```
unzipTemplate("chapterTemplate.pdf");
```

Open the chapterTemplate unzipped folder.



In an HTML or text editor, open the docpart\_templates.html file.

```

<dptemplate name="Section1">
  <layout style="page-margin:0.5in 1in 0.5in 1in 0.5in 0in; page-size: 8.5in 11in portrait">
    <pheader type="first" template-name="SectionFirstPageHeader"/>
    <pfooter type="first" template-name="SectionFirstPageFooter"/>
    <pheader type="default" template-name="SectionDefaultPageHeader"/>
    <pfooter type="default" template-name="SectionDefaultPageFooter"/>
    <pheader type="even" template-name="SectionEvenPageHeader"/>
    <pfooter type="even" template-name="SectionEvenPageFooter"/>
    <pnumberformat="1" />
  </layout>
  <hole id="Title" default-style-name="SectionTitle1">TITLE</hole>
  <hole id="Content" default-style-name="SectionContent">CONTENT</hole>
</dptemplate>

```

To modify the page size and margins to A4 values, modify the `page-margin` and `page-size` properties of the `style` attribute of the `layout` element.

Set the `page-margin` property value to:

```
0.98in 0.98in 0.98in 0.98in 0.5in 0.5in 0in
```

This value specifies the margins in this order:

- 1 Top margin of 0.98 inches.
- 2 Left margin of 0.98 inches
- 3 Bottom margin of 0.98 inches
- 4 Right margin of 0.98 inches
- 5 Header of 0.5 inches
- 6 Footer of 0.5 inches
- 7 Gutter size of 0

Set the `page-size` property to:

```
8.27in 11.69in portrait
```

This value specifies an 8.27-by-11.69 inch page size in portrait orientation.

The template now looks like this:



```

<dptemplate name="Section1">
  <layout style="page-margin:0.98in 0.98in 0.98in 0.98in 0.5in 0.5in 0in; page-size:8.27in 11.69in portrait">
    <pheader type="first" template-name="SectionFirstPageHeader"/>
    <pfooter type="first" template-name="SectionFirstPageFooter"/>
    <pheader type="default" template-name="SectionDefaultPageHeader"/>
    <pfooter type="default" template-name="SectionDefaultPageFooter"/>
    <pheader type="even" template-name="SectionEvenPageHeader"/>
    <pfooter type="even" template-name="SectionEvenPageFooter"/>
    <pnumber format="1" />
  </layout>
  <hole id="Title" default-style-name="SectionTitle1">TITLE</hole>
  <hole id="Content" default-style-name="SectionContent">CONTENT</hole>
</dptemplate>

```

Save the `docpart_templates.html` file.

Zip the template.

```
zipTemplate("chapterTemplate.pdf");
```

### Use the Customized Templates in a Report

Use the customized template to create a PDF report on magic squares, that is, matrices whose columns, rows, and diagonals each add up to the same number.

### Import DOM and Report API Packages

Import the DOM and Report API packages to avoid having to use fully qualified class names.

```
import mlreportgen.report.*;
import mlreportgen.dom.*;
```

Create a report container.

```
rpt = Report("report", "pdf");
```

### Add a Title Page

Create a title page and specify the title, subtitle, and author.

```
title = TitlePage("Title", "Magic Squares");
title.Subtitle = "Columns, Rows, Diagonals: All Equal Sums";
title.Author = "Albrecht Durer";
```

Use the custom title page template to generate the title page.

```
title.TemplateSrc = fullfile(pwd, "titlePageTemplate.pdf");
```

Add the title page to the report.

```
append(rpt, title);
```

### Add a Table of Contents

```
toc = TableOfContents;
```

Use the custom table of contents template to generate the table of contents.

```
toc.TemplateSrc = fullfile(pwd,"tableOfContentsTemplate.pdf");
```

Add the table of contents to the report.

```
append(rpt,toc);
```

### **Add a Chapter**

Create a chapter and specify the title.

```
chapter = Chapter("Introduction");
```

Use the custom chapter template to generate the chapter.

```
chapter.TemplateSrc = fullfile(pwd,"chapterTemplate.pdf");
```

Create a section and add a paragraph to it.

```
sec1 = Section("What is a Magic Square?");  
para = Paragraph(['A magic square is an N-by-N matrix '...  
'constructed from the integers 1 through N^2 '...  
'with equal row, column, and diagonal sums.']);  
append(sec1,para);
```

Add the section to the chapter.

```
append(chapter,sec1);
```

Create another section and add a paragraph to it.

```
sec2 = Section("Albrecht Durer and the Magic Square");  
para = Paragraph([ ...  
'The German artist Albrecht Durer (1471-1528) created '...  
'many woodcuts and prints with religious and '...  
'scientific symbolism. One of his most famous works, '...  
'Melancholia I, explores the depressed state of mind '...  
'which opposes inspiration and expression. '...  
'Renaissance astrologers believed that the Jupiter '...  
'magic square (shown in the upper right portion of '...  
'the image) could aid in the cure of melancholy. The '...  
'engraving's date (1514) can be found in the '...  
'lower row of numbers in the square.']);  
append(sec2,para);
```

Add the section to the chapter.

```
append(chapter,sec2);
```

Add the chapter to the report.

```
append(rpt,chapter);
```

### **Close and View the Document**

```
close(rpt);  
rptview(rpt);
```

### **See Also**

`mlreportgen.report.TitlePage` | `mlreportgen.report.TableOfContents` |  
`mlreportgen.report.Chapter`

### **More About**

- “Customize the Page Size and Margins of a Word Report Using Templates” on page 17-135
- “Customize the Page Size and Margins of a Report Programmatically” on page 17-154
- “Report Formatting Approaches” on page 13-17

## Customize the Page Size and Margins of a Report Programmatically

This example shows how to use a programmatic approach to customize the page size and the page margins, header, footer, and gutter in a PDF or Microsoft® Word report. Alternatively, you can use a template-based approach. For examples that use the template-based approach, see “Customize the Page Size and Margins of a Word Report Using Templates” on page 17-135 and “Customize the Page Size and Margins of a PDF Report Using Templates” on page 17-145. Use the template-based approach if you plan to make other modifications to the templates. Otherwise, use the programmatic approach.

By default, the Report API generates pages that have an 11-inch height, 8.5-inch width, and 1-inch margins. Headers and footers are each 0.5 inch wide and the gutter is 0 pixels. You might need to change these dimensions in your report. For example, if you are generating a report for a European locale, you might need to generate A4 pages, which have an 11.69-inch height, 8.27-inch width, and 0.98-inch margins.

To customize the page dimensions of a Report API report programmatically, assign custom `mlreportgen.dom.PageSize` and `mlreportgen.dom.PageMargins` objects to the layout object used by the whole report or a report section. The layout object for the whole report is an `mlreportgen.report.ReportLayout` object, which is assigned to the `Layout` property of the `mlreportgen.report.Report` object. The layout for a report section is an `mlreportgen.report.ReporterLayout` object, which is assigned to the `Layout` property of the reporter object that represents the section. Objects of these reporter classes have a `Layout` property that you can use to customize the page dimensions for the section:

- `mlreportgen.report.TitlePage`
- `mlreportgen.report.TableOfContents`
- `mlreportgen.report.ListOfFigures`
- `mlreportgen.report.ListOfTables`
- `mlreportgen.report.ListOfCaptions`
- `mlreportgen.report.Chapter`

If you customize the page dimensions for the whole report and then override the dimensions for a report section, specify all of the properties of the `PageSize` or `PageMargins` objects that you create for the section. Otherwise, the properties that you do not specify have default values, which might be different from the values that you specified for the whole report.

In this example, you create a report that has A4 page dimensions and then override those dimensions for the title page.

### Create the Report Container

Import the DOM and Report API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*;
import mlreportgen.report.*;
```

Create a report container for a Word report. For a PDF report, replace "docx" with "pdf".

```
rpt = mlreportgen.report.Report("myreport", "docx");
```

### Specify Custom Page Dimensions for the Whole Report

Create a `PageSize` object that specifies an 11.69-inch page height, 8.27-inch page width, and portrait orientation. Assign the `PageSize` object to the report layout object.

```
pageSizeObj = PageSize("11.69in", "8.27in", "portrait");
rpt.Layout.PageSize = pageSizeObj;
```

Create a `PageMargins` object that specifies

- Top, bottom, left, right margins of 0.98 inches
- Header and footer heights of 0.5 inches
- Gutter size of 0 inches

```
pageMarginsObj = PageMargins();
pageMarginsObj.Top = "0.98in";
pageMarginsObj.Bottom = "0.98in";
pageMarginsObj.Left = "0.98in";
pageMarginsObj.Right = "0.98in";
pageMarginsObj.Header = "0.5in";
pageMarginsObj.Footer = "0.5in";
pageMarginsObj.Gutter = "0in";
```

Assign the `PageMargins` object to the report layout object.

```
rpt.Layout.PageMargins = pageMarginsObj;
```

### Override the Page Dimensions for the Title Page Section

Create a title page and specify 2-inch margins, instead of the 0.98 margins specified for the whole report. Specify a 0.5-inch header and footer and a 0-inch gutter.

```
title = TitlePage("Title", "Magic Squares");
title.Subtitle = "Columns, Rows, Diagonals: All Equal Sums";
title.Author = "Albrecht Durer";
pageMarginsObj = PageMargins();
pageMarginsObj.Top = "2in";
pageMarginsObj.Bottom = "2in";
pageMarginsObj.Left = "2in";
pageMarginsObj.Right = "2in";
pageMarginsObj.Header = "0.5in";
pageMarginsObj.Footer = "0.5in";
pageMarginsObj.Gutter = "0in";
```

Assign the `PageMargins` object to the `TitlePage` layout object.

```
title.Layout.PageMargins = pageMarginsObj;
append(rpt, title);
```

The page size will be the size specified for the whole report because you did not assign a `PageSize` object to the `TitlePage` object layout.

### Create the Rest of the Report Using the Page Dimensions Specified for the Whole Report

Create the table of contents section. Do not specify a custom page size or custom margins. The page dimensions will be the dimensions specified for the whole report.

```
toc = TableOfContents;  
append(rpt,toc);
```

Create a chapter that uses the default page dimensions.

```
chapter = Chapter("Introduction");  
  
sec1 = Section("What is a Magic Square?");  
para = Paragraph(['A magic square is an N-by-N matrix '...  
'constructed from the integers 1 through N^2 '...  
'with equal row, column, and diagonal sums.']);  
append(sec1,para);  
  
append(chapter,sec1);  
sec2 = Section("Albrecht Durer and the Magic Square");  
para = Paragraph(['...  
'The German artist Albrecht Durer (1471-1528) created '...  
'many woodcuts and prints with religious and '...  
'scientific symbolism. One of his most famous works, '...  
'Melancholia I, explores the depressed state of mind '...  
'which opposes inspiration and expression. '...  
'Renaissance astrologers believed that the Jupiter '...  
'magic square (shown in the upper right portion of '...  
'the image) could aid in the cure of melancholy. The '...  
'engraving's date (1514) can be found in the '...  
'lower row of numbers in the square.']);  
append(sec2,para);  
  
append(chapter,sec2);  
  
append(rpt,chapter);
```

### Close and View the Report

```
close(rpt);  
rptview(rpt);
```

### See Also

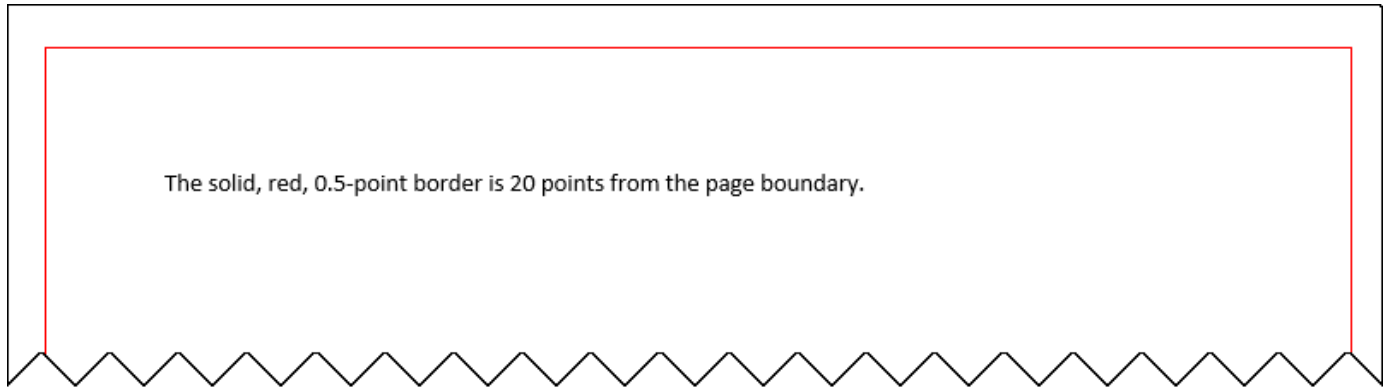
`mlreportgen.report.ReportLayout` | `mlreportgen.report.ReporterLayout` |  
`mlreportgen.report.Report` | `mlreportgen.report.Reporter` |  
`mlreportgen.dom.PageSize` | `mlreportgen.dom.PageMargins`

### More About

- “What Is a Reporter?” on page 1-3
- “Customize the Page Size and Margins of a Word Report Using Templates” on page 17-135
- “Customize the Page Size and Margins of a PDF Report Using Templates” on page 17-145

## Generate a Microsoft Word Document with Page Borders

This example generates a Microsoft Word document that has solid, red, 0.5-point borders positioned 20 points from the page boundary.



Import the DOM API package so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*;
```

Create a Word document.

```
d = Document("myDocument", "docx");
open(d);
```

Create a `PageBorder` object and specify the border style, color, and width.

```
pageBorder = PageBorder("solid", "red", "0.5pt");
```

Specify that the border position is relative to the page boundary and specify the margin between the page boundary and the border.

```
pageBorder.MeasureFrom = "pageboundary";
pageBorder.Margin = "20pt";
```

Set the `PageBorder` property of the layout associated with the document to the `PageBorder` object.

```
d.CurrentPageLayout.PageBorder = pageBorder;
```

Add text to the document. Close and view the document.

```
append(d, "The solid, red, 0.5-point border is 20 points from the page boundary.");
close(d);
rptview(d);
```

### See Also

`mlreportgen.dom.PageBorder` | `mlreportgen.dom.DOCXPageLayout` |  
`mlreportgen.dom.PDFPageLayout` | `mlreportgen.report.ReporterLayout` |  
`mlreportgen.report.ReportLayout`

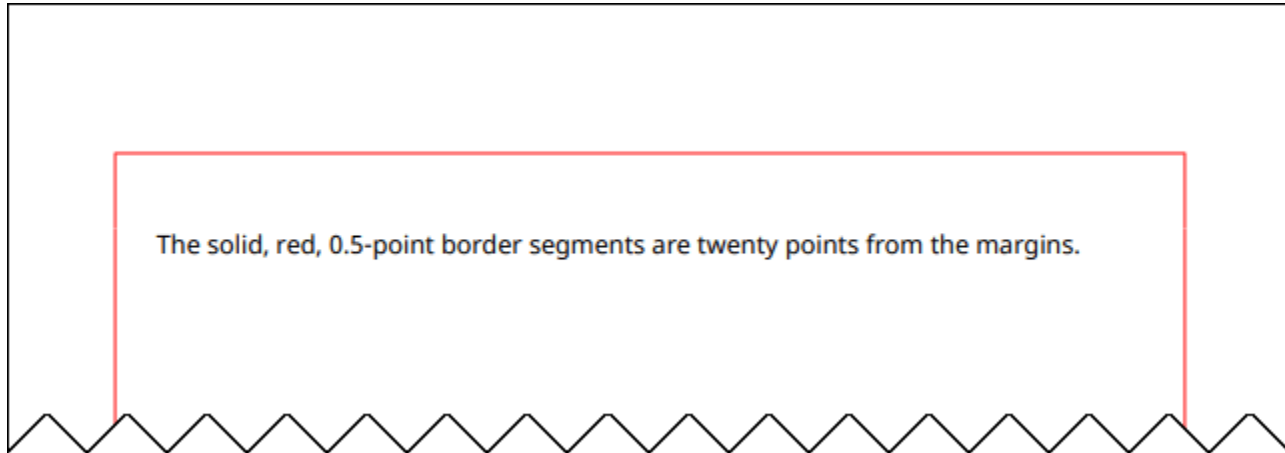
## **Related Examples**

- “Generate a PDF Document with Page Borders” on page 17-159
- “Generate a Report API Report with Page Borders” on page 17-161
- “Create Page Layout Sections” on page 13-144
- “Create Page Footers and Headers” on page 13-148



## Generate a PDF Document with Page Borders

This example generates a PDF document that has solid, red, 0.5-point borders positioned 20 points from the page margins.



Import the DOM API package so that you do not have to use long, fully qualified class names.

```
import mlreportgen.dom.*;
```

Create a PDF document.

```
d = Document("myDocument", "pdf");
open(d);
```

Create a `PageBorder` object and specify the style, color, width, and margin for all border segments. For PDF Documents, the margin of a border segment specifies the distance between the segment and the page margin.

```
pageBorder = PageBorder("solid", "red", "0.5pt", "20pt");
```

Set the `PageBorder` property of the layout associated with the document to the `PageBorder` object.

```
d.CurrentPageLayout.PageBorder = pageBorder;
```

Add text to the document. Close and view the document.

```
append(d, "The solid, red, 0.5-point border segments are twenty points from the margins.");
close(d);
rptview(d);
```

### See Also

`mlreportgen.dom.PageBorder` | `mlreportgen.dom.DOCXPageLayout` |  
`mlreportgen.dom.PDFPageLayout` | `mlreportgen.report.ReporterLayout` |  
`mlreportgen.report.ReportLayout`

### Related Examples

- “Generate a Microsoft Word Document with Page Borders” on page 17-157

- “Generate a Report API Report with Page Borders” on page 17-161
- “Create Page Layout Sections” on page 13-144
- “Create Page Footers and Headers” on page 13-148

## Generate a Report API Report with Page Borders

This example uses the Report API to generate a Word or PDF report that has red page borders on the title page and blue page borders on the other sections of the report.

Import the Report API and DOM API packages so that you do not have to use long, fully qualified class names.

```
import mlreportgen.report.*;
import mlreportgen.dom.*;
```

Create a Word report and specify solid, blue, 0.5-point default borders for all pages of the report. For a PDF report, replace "docx" with "pdf".

```
rpt = Report("myreport", "docx");
rpt.Layout.PageBorder = PageBorder("solid", "blue", "0.5pt");
```

Create a title page and specify solid, red, 0.5-point borders for the title page. Append the title page to the report.

```
tp = TitlePage("Title", "My Report");
tp.Layout.PageBorder = PageBorder("solid", "red", "0.5pt");
append(rpt, tp);
```

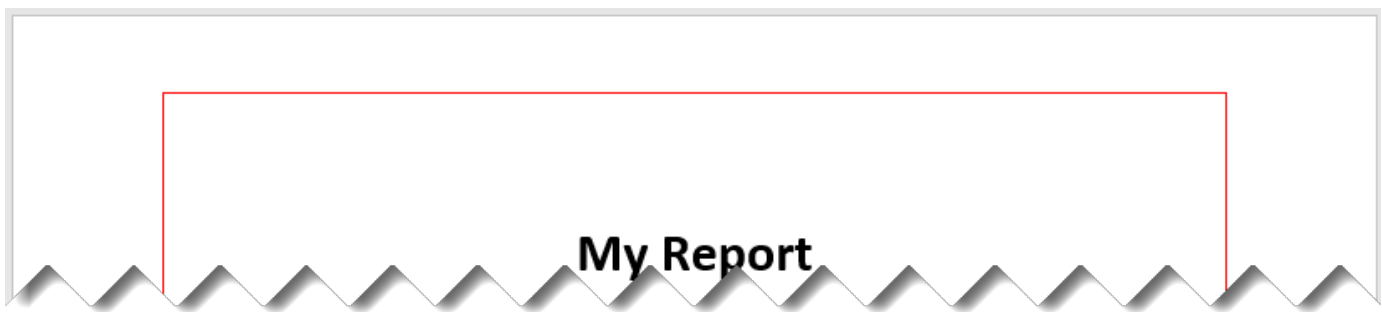
Create a chapter and append it to the report.

```
ch = Chapter("My Chapter");
append(rpt, ch);
```

Close and view the report.

```
close(rpt);
rptview(rpt);
```

In the generated report, the title page has red borders as specified by the title page reporter layout.



The chapter pages have the default blue borders because the chapter reporter layout did not specify page borders.



### See Also

`mlreportgen.dom.PageBorder` | `mlreportgen.dom.DOCXPageLayout` |  
`mlreportgen.dom.PDFPageLayout` | `mlreportgen.report.ReporterLayout` |  
`mlreportgen.report.ReportLayout`

### Related Examples

- “Generate a Microsoft Word Document with Page Borders” on page 17-157
- “Generate a PDF Document with Page Borders” on page 17-159
- “Create Page Layout Sections” on page 13-144
- “Create Page Footers and Headers” on page 13-148