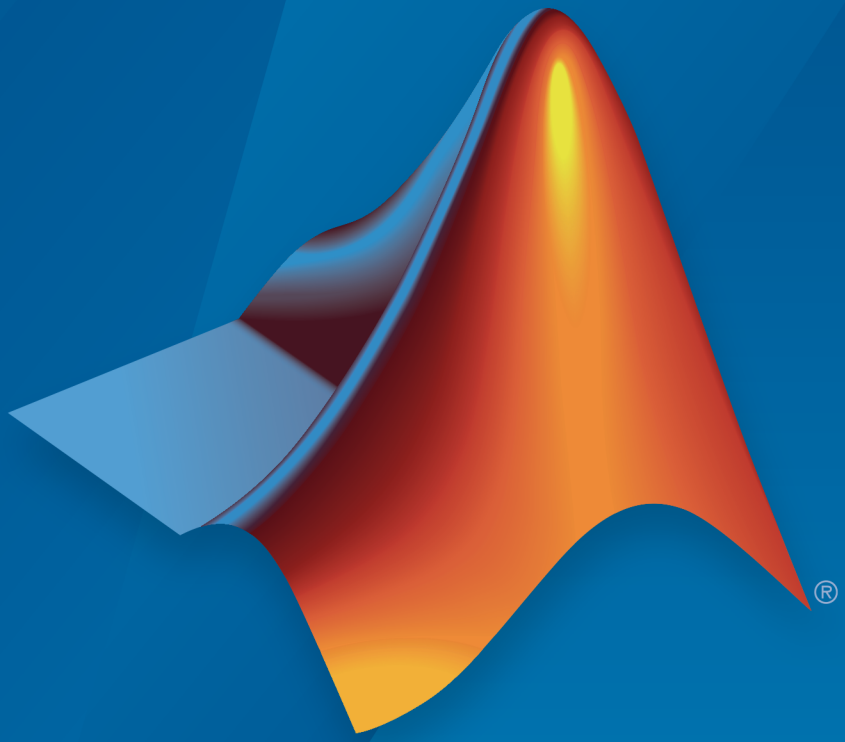
MATLAB® Report Generator™

User's Guide



MATLAB®

R2016a

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

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

MATLAB® Report Generator™ User's Guide

© COPYRIGHT 1999–2016 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 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 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)

Contents

1 Getting Started

MATLAB Report Generator Product Description 1-2
Key Features 1-2

MATLAB Code and Results Presentation 1-3

Report Creation Workflow 1-4
Approaches for Creating Reports 1-4
Interactive Report Generation 1-4

How MATLAB Report Generator and MATLAB Software Interact 1-6

Report Components 1-7
Types of Report Components 1-7

Working with the Report Explorer 1-8
About the Report Explorer 1-8

2 Create Your First Report

Create a MATLAB Report 2-2

Create a Report Setup File 2-3

Add Report Content Using Components 2-5
Report Components 2-5
Specify Report Variables 2-7
Create a Title Page 2-9

Add a Chapter 2-12
Add Introductory Text to the First Chapter 2-14
Add an Image 2-16
Create the Magic Squares and Their Images 2-21
Create a For Loop 2-22
Add a Chapter for Each Square 2-24
Determine the Matrix Size 2-26
Insert the Magic Square Size into the Report 2-28
Create the Magic Square 2-29
Add Display Logic 2-32
Display the Magic Square 2-34

Generate a Report 2-39

3 Set Up a Report

Report Setups 3-2
Setup Hierarchy 3-2
Setup Files 3-2
Create a Report Setup 3-3

Create a New Setup File 3-4
Create Setup File Using the Report Explorer 3-4
Create Setup File Programmatically 3-4
Working with Setup Files 3-4

Open a Report Setup 3-6
Opening a Setup on the MATLAB Path 3-6
Opening a Setup Not on the MATLAB Path 3-7
Opening a Setup Programmatically 3-7

Close a Report Setup 3-8
Close a Setup Using the Report Explorer 3-8
Close a Setup Programmatically 3-8

Save a Report Setup 3-9
Save a Setup Under Its Existing Name 3-9
Save a Setup Under a New Name 3-9

Load Report Setup into MATLAB Workspace 3-10

Insert Components 3-11
Point-and-Click Method 3-11
Drag-and-Drop Method 3-11
Fix Context Violations 3-11

Set Component Properties 3-12
Edit Component Property Values 3-12
Computed Property Values 3-12

Move Components 3-13
Point-and-Click Method 3-13
Drag-and-Drop Method 3-14

Delete Components 3-15

Deactivate Components 3-16

Send Components to the MATLAB Workspace 3-17

4 Generate a Report

Generate a Report 4-2
Run a Report 4-2
Report Output Options 4-2

Select Report Generation Options 4-4
Report Options Dialog Box 4-4
Report Output Format 4-4
PDF Stylesheets 4-7
Web Stylesheets 4-7
RTF (DSSSL Print) and Word Stylesheets 4-8
Report Generation Processing 4-8
Location of Report Output File 4-10
Report Description 4-12

Report Generation Preferences 4-13
Report Generator Preferences Pane 4-13

File Format and Extension 4-14
Image Formats 4-15
Report Viewing 4-15
Reset to Defaults 4-16
Edit HTML Command 4-16

Change Report Locale 4-17

Convert XML Documents to Different File Formats 4-18
Why Convert XML Documents? 4-18
Convert XML Documents Using the Report Explorer 4-18
Convert XML Documents Using the Command Line 4-20
Edit XML Source Files 4-20

Create a Report Log File 4-21

Generate MATLAB Code from Report Setup File 4-22

Troubleshooting Report Generation Issues 4-25
Memory Usage 4-25
HTML Report Display on UNIX Systems 4-26

5 Add Content with Components

Components 5-2
Component Formatting 5-3

Report Structure Components 5-4

Table Formatting Components 5-5

Property Table Components 5-6
About Property Table Components 5-6
Open the Example Report Template 5-8
Examine the Property Table Output 5-8
Select Object Types 5-9
Display Property Name/Property Value Pairs 5-9
Edit Table Titles 5-12
Enter Text into Table Cells 5-12

Add, Replace, and Delete Properties in Tables 5-13
Format Table Columns, Rows, and Cells 5-14
Zoom and Scroll 5-16
Select a Table 5-16

Summary Table Components 5-17
About Summary Table Components 5-17
Open the Example Report Template 5-18
Select Object Types 5-19
Add and Remove Properties 5-19
Set Relative Column Widths 5-20
Set Object Row Options 5-20

Logical and Looping Components 5-21

Edit Figure Loop Components 5-22
Figure Loop in a Report 5-22
Figure Properties 5-23
Loop on the Current Figure 5-24
Loop on Visible Figures 5-24
Loop on Figures with Tags 5-24
Modify Loop Section Options 5-24

6 Template-Based Report Formatting

Report Conversion Templates 6-2
Templates for Report Conversion 6-2
Customizing Templates 6-2

Generate a Report Using a Template 6-4

Conversion Template Contents 6-5
Default Styles 6-5
Part Templates 6-9
Header and Footers in Word Conversion Templates 6-10

Copy a Conversion Template 6-12
Copy a Conversion Template 6-12

Open a Conversion Template **6-13**

Set Conversion Template Properties **6-14**

Move a Conversion Template **6-15**

Delete a Conversion Template **6-16**

Customize Microsoft Word Report Styles **6-17**
Customize Default Microsoft Word Component Styles **6-17**
Create Styles in a Microsoft Word Template **6-17**

Customize Microsoft Word Part Templates **6-20**
Custom Word Part Templates **6-20**
Display the Developer Ribbon in Word **6-21**
Customize a Word Conversion Part Template **6-21**
Set Default Text Style for a Hole **6-22**
Distinguish Inline and Block Holes **6-24**
Avoid Changing Block Holes to Inline Holes **6-25**
Delete a Hole **6-25**
Add an Inline Hole **6-27**
Add a Block Hole **6-28**
Remove or Modify Chapter Prefix **6-28**

Customize a Microsoft Word Title Page Template **6-30**
Create a Custom Template **6-30**
Change the Color of a Report Title **6-31**
Assign the Template to a Report **6-33**
Customize Title Page Content and Layout **6-34**

Create a Custom HTML or PDF Template **6-36**
Copy the Template **6-36**
Assign the Template to a Report **6-36**
Select an HTML Editor **6-37**
Edit HTML or PDF Templates **6-38**
Edit HTML or PDF Styles in a Template **6-38**

7 Create Custom Components

Create Custom Components 7-2

Define Components 7-5
Required Component Data 7-5
Specify the Location of Component Files 7-5
Set Component Display Options 7-6
Specify Component Properties 7-8
Modify Existing Components 7-10
Build Components 7-10
Rebuild Existing Components 7-11
Remove a Component 7-11

Specify Tasks for a Component to Perform 7-12
About Component Customization 7-12
Required Customization: Specify Format and Content of Report Output 7-12
Change a Component's Outline String in the Report Explorer Hierarchy 7-14
Modify the Appearance of Properties Dialog Boxes 7-15
Specify Additional Component Properties 7-16

Customized Components 7-18
Fetching Securities Data and Displaying It in a Table 7-18
Displaying Securities Data in Two Tables 7-23

8 Create Custom Stylesheets

Stylesheets 8-2
Built-In Versus Custom Stylesheets 8-2
Customize Stylesheets Using Data Items 8-3

Create a New Stylesheet 8-4

Edit, Save, or Delete a Stylesheet 8-5
Edit a Stylesheet 8-5

Save a Stylesheet 8-7
Delete a Stylesheet 8-8

Edit Stylesheet Data Items 8-9
Data Item Categories in Built-In Stylesheets 8-9
Edit Data Items in Simple or Advanced Edit Mode 8-13
Data Items 8-13

Stylesheet Cells for Headers and Footers 8-23
About Stylesheet Cells and Cell Groups 8-23
Headers and Footers 8-24
Add Content to Headers and Footers Using Templates 8-26
Insert Graphics Files 8-27
Modify Fonts and Other Properties 8-27

Customized Stylesheets 8-29
Number Pages in a Report 8-29
Add Graphics to Headers in PDF Reports 8-30
Change Font Size, Page Orientation, and Paper Type of a Generated Report 8-35
Edit Font Size as a Derived Value in XML 8-37

Configure PDF Fonts 8-40
Use Additional Fonts for PDF Output 8-40
PDF Font Support for Languages 8-41
Identifying When to Specify a Font 8-42
Stylesheets Override PDF Font Mapping 8-42
Non-English PDF Font Mapping Tasks 8-42
lang_font_map.xml File 8-42
Locate Non-English Fonts 8-44
Add or Modify Language Font Mappings 8-46
Specify the Location of Font Files 8-46

9 Comparing XML Files

Compare XML Files 9-2

How to Compare XML Files 9-4
Select Files to Compare 9-4

Change Comparison Type 9-5
XML Comparison Examples 9-5
See Also 9-5

Explore the XML Comparison Report 9-6
Navigate the XML Comparison Report 9-6
Save Comparison Log Files in a Zip File 9-8
Export Results to the Workspace 9-8

How the Matching Algorithm Works 9-10
How the Chawathe Algorithm Works 9-10
Why Use a Heuristic Algorithm? 9-12
Examples of Matching Results 9-12

10 Components — Alphabetical List

11 Functions – Alphabetical List

12 Classes – Alphabetical List

13 Create a Report Program

Create a Report Program 13-3

Document Object Model 13-4
DOM Object Help and Documentation 13-4

Construct a DOM Object **13-5**

Import the DOM API Package **13-6**

Use Dot Notation for DOM Object Properties **13-7**

Create a Document Object to Hold Content **13-9**

Add Content to a Report **13-11**

Clone a DOM Object **13-13**

Add Content as a Group **13-14**

Stream a Report **13-16**

Report Packages **13-17**

Close a Report **13-18**

Display a Report **13-19**

Report Formatting Approaches **13-21**
Style Sheets in Templates **13-21**
Format Objects **13-21**
Format Properties **13-22**

Use Style Sheet Styles **13-24**

Format Inheritance **13-26**

Templates for DOM API Report Programs **13-27**
Template Packages **13-27**
Styles **13-28**
Page Layout **13-28**
Document Part Templates **13-28**

Create Object Containers **13-30**

Form-Based Reporting **13-32**

Fill the Blanks in a Report Form **13-33**

Use Subforms in a Report **13-35**

Create a Microsoft Word Document Part Template Library **13-37**
Create Document Part Template Library in a Word Template **13-37**

Create an HTML Document Part Template Library **13-40**
HTML Document Part Template Library Structure **13-40**

Create a PDF Document Part Template Library **13-42**
PDF Document Part Template Library Structure **13-42**
Document Part Template Library Contents **13-43**

Object-Oriented Report Creation **13-48**

Simplify Filling in Forms **13-49**

Create and Format Text **13-51**
Create Text **13-51**
Create Special Characters **13-51**
Append HTML or XML Markup **13-52**
Format Text **13-52**

Create and Format Paragraphs **13-57**
Create a Paragraph **13-57**
Create a Heading **13-57**
Format a Paragraph **13-57**

Create and Format Lists **13-63**
Create an Ordered or Unordered List **13-63**
Create a Multilevel List **13-65**
Format Lists **13-66**

Create and Format Tables **13-69**
Two Types of Tables **13-69**
Create a Table from a Two-Dimensional Array **13-70**
Create a Table Using the Table entry Function **13-70**
Create a Table from Scratch **13-71**
Format a Table **13-72**
Create a Formal Table **13-77**
Format a Formal Table **13-77**
Create and Format Table Rows **13-78**

Format Table Columns 13-79
Create and Format Table Entries 13-80

Create Links 13-83
Create a Link Target 13-83
Create an External Link 13-83
Create an Internal Link 13-84
Add Text or Images to Links 13-84
Create a Page Reference 13-84

Create and Format Images 13-86
Create an Image 13-86
Resize an Image 13-87
Image Storage 13-87
Links from an Image 13-87

Create a Table of Contents 13-88
Create a TOC in Your Report Program 13-88
Use a Template to Create a Microsoft Word TOC 13-89
Create Table of Contents in HTML or PDF Templates 13-91
Set Outline Levels of Section Heads 13-93

Create Image Maps 13-96

Automatically Number Document Content 13-98
Automatically Number Content Programmatically 13-98
Automatically Number Content Using Part Templates . . 13-100

Appending HTML to DOM Reports 13-103
Workflow for Appending HTML 13-103

Append HTML Content to DOM Reports 13-105
Use an addHTML Method 13-105
Append an HTML Object 13-106
Address Errors 13-106

Append HTML File Contents to DOM Reports 13-107
Use an addHTMLFile Method 13-107
Append an HTMLFile Object 13-107
Address Errors 13-108

Use an HTML Cleanup Program 13-109
Use HTML Tidy to Fix HTML Code 13-109

HTML Code Requirements for DOM Reports **13-113**
XML-Parsable HTML Code **13-113**
Supported HTML Elements and Attributes **13-113**
Supported HTML CSS Style Attributes for All Elements . **13-115**
Support for HTML Character Entities **13-117**
DOCTYPE Declaration **13-117**

Display Report Generation Messages **13-119**
Report Generation Messages **13-119**
Display DOM Default Messages **13-119**
Create and Display a Progress Message **13-121**

Compile a Report Program **13-123**

Create a Microsoft Word Template **13-124**

Add Holes in a Microsoft Word Template **13-125**
Display the Developer Ribbon in Word **13-125**
Inline and Block Holes **13-125**
Create an Inline Hole **13-126**
Create a Block-Level Hole **13-126**
Set Default Text Style for a Hole **13-127**

Modify Styles in a Microsoft Word Template **13-130**
Edit Styles in a Word Template **13-130**
Add Styles to a Word Template **13-131**

Create an HTML or PDF Template **13-135**
Edit an HTML or PDF Template **13-135**

Add Holes in HTML and PDF Templates **13-137**
Types of Holes **13-137**
Create a Hole **13-137**

PDF and HTML Document Parts and Holes **13-139**
Add Template to PDF Document Part Template Library . **13-139**
Use the Document Part Template in a Report Program . . **13-141**

Modify Styles in HTML Templates **13-143**

Modify Styles in PDF Templates **13-144**
PDF Style Sheets **13-144**

Create Page Layout Sections **13-148**
Define Page Layouts in a Word Template **13-148**
Define Page Layouts in a PDF Template **13-148**
Navigate Template-Defined Page Layouts **13-150**
Override Template Page Layouts in Your Report Program **13-150**
Create Layouts Programmatically **13-151**

Create Page Footers and Headers **13-152**
Use Page Headers and Footers in a Template **13-152**
Create Running Page Headers and Footers **13-157**
Create Page Headers and Footers Programmatically **13-159**

Add Complex Page Numbers in Microsoft Word **13-162**

14 Programmatic PowerPoint Presentation Creation

Create a Presentation Program **14-2**
PPT API Programs **14-3**
Two Ways to Use the PPT API **14-4**
PPT API Applications and PowerPoint Templates **14-5**
Template Elements **14-5**

Create PPT Objects **14-7**
PPT Objects **14-7**
Use a PPT Constructor **14-7**
PPT Objects Created Without Constructors **14-8**

Import the PPT API Package **14-10**

Get and Set PPT Object Properties **14-11**

Create a Presentation Object to Hold Content **14-13**

Generate a Presentation **14-14**

Display Presentation Generation Messages **14-15**
Presentation Generation Messages **14-15**
Display PPT Default Messages **14-15**
Create and Display a Progress Message **14-17**

Compile a Presentation Program **14-19**

Presentation Formatting Approaches **14-20**
Template Formatting **14-21**
Format Objects **14-21**
Format Properties **14-22**
Interactive Formatting of Slide Content **14-22**

Presentation Format Inheritance **14-24**

Set Up a PowerPoint Template **14-26**
Use Existing Presentations as Templates **14-26**
Customize a Copy of the Default Template **14-26**
Global Presentation Formatting Using a Slide Master **14-27**
Add a Slide Master **14-28**
Format a Slide Layout **14-30**
Add a Slide Layout **14-32**
Add a Placeholder **14-33**

Access PowerPoint Template Elements **14-37**
PPT API Applications and PowerPoint Templates **14-5**
Template Elements **14-5**
View and Change Slide Master Names **14-38**
View and Change Slide Layout Names **14-39**
View and Change Placeholder and Content Object Names . **14-41**

Define a Style Using Format Objects **14-43**

Use Format Properties **14-45**
Dot Notation **14-45**
Get the Properties of an Object **14-45**
Set the Properties of an Object **14-46**

Update Presentation Content Programmatically **14-48**
Generate the Existing Presentation **14-48**
Updates to the Presentation **14-50**
Set Up the Existing Presentation **14-52**
Import the PPT API Package **14-53**
Create the `Presentation` Object **14-53**
Replace a Picture **14-53**
Replace Text with Links **14-54**
Replace a Table **14-54**
Insert a New Slide **14-55**

Generate and Open the Presentation 14-55
Code for `myUpdatedPresentation` 14-56

Create a Presentation Programmatically 14-59
Set Up a Template 14-61
Import the PPT API Package 14-63
Create the `Presentation` Object 14-63
Add a Presentation Title Slide 14-64
Add a Slide with a Picture 14-65
Add a Slide with Text 14-65
Add a Slide with a Table 14-66
Generate and Open the Presentation 14-67
Code for `myNewPPTPresentation` 14-68

Add Slides 14-71
Specify the Order of a Slide 14-71
Specify the Slide Master 14-73

Add and Replace Presentation Content 14-74
Set Up the Template 14-74
Replace Content 14-75
Add and Replace Text 14-75
Add or Replace a Table 14-78
Add or Replace a Picture 14-79

Create and Format Text 14-81
Create Text 14-81
Create a Subscript or Superscript 14-81
Format Text 14-82

Create and Format Paragraphs 14-84
Create a Paragraph 14-84
Format Paragraph Content 14-84

Create and Format Tables 14-87
Create a Table 14-87
Format a Table 14-87
View Table Style Names 14-93

Create and Format Pictures 14-96
Create a Picture 14-96
Format a Picture 14-97

Create and Format Links **14-98**
Create an External Link **14-98**
Format an External Link **14-98**

15 Classes Being Removed

1

Getting Started

- "MATLAB Report Generator Product Description" on page 1-2
- "MATLAB Code and Results Presentation" on page 1-3
- "Report Creation Workflow" on page 1-4
- "How MATLAB Report Generator and MATLAB Software Interact" on page 1-6
- "Report Components" on page 1-7
- "Working with the Report Explorer" on page 1-8

MATLAB Report Generator Product Description

Design and generate reports from MATLAB applications

MATLAB Report Generator lets you create richly formatted Microsoft® Word, Microsoft PowerPoint, HTML, and PDF reports that present results from your MATLAB programs and applications. You can use the prebuilt, customizable Word and HTML templates to lay out and format reports. You can also design and create reports based on your organization's templates and standards.

MATLAB Report Generator automatically captures results and figures across multiple MATLAB functions and presents them within a single report.

Key Features

- Automated reporting from MATLAB
- Report formatting based on Microsoft Word and HTML templates
- Report designer app for creating custom Word, HTML, and PDF reports
- Selective report generation via logical control flow components, such as IF, THEN, ELSE, and WHILE
- API for forms-based Word, PowerPoint, HTML and PDF report generation

MATLAB Code and Results Presentation

You can use the MATLAB Report Generator to create reports for sharing your MATLAB code and presenting the results of the code.

In addition, MATLAB provides several methods for presenting MATLAB code and results, including:

- MATLAB `publish` command
- MATLAB `notebook` command

MATLAB enables you to publish your MATLAB code quickly, so that you can describe and share your code with others, even if they do not have MATLAB software. You can publish in various formats, including HTML, XML, and LaTeX. If Microsoft Word or Microsoft PowerPoint® applications are on your Microsoft Windows® system, you can publish to their formats as well.

On Windows platform, you can use the `notebook` command to create a Microsoft Word document that contains text, MATLAB commands, and the output from MATLAB commands. The document is a record of an interactive MATLAB session annotated with text or a document embedded with live MATLAB commands and output.

To compare the MATLAB tools for presenting MATLAB code and results and MATLAB Report Generator, see “Options for Presenting Your Code”.

Report Creation Workflow

Approaches for Creating Reports

You can create and generate reports :

- Interactively, using the Report Explorer
- Programmatically, using the DOM (Document Object Model) API

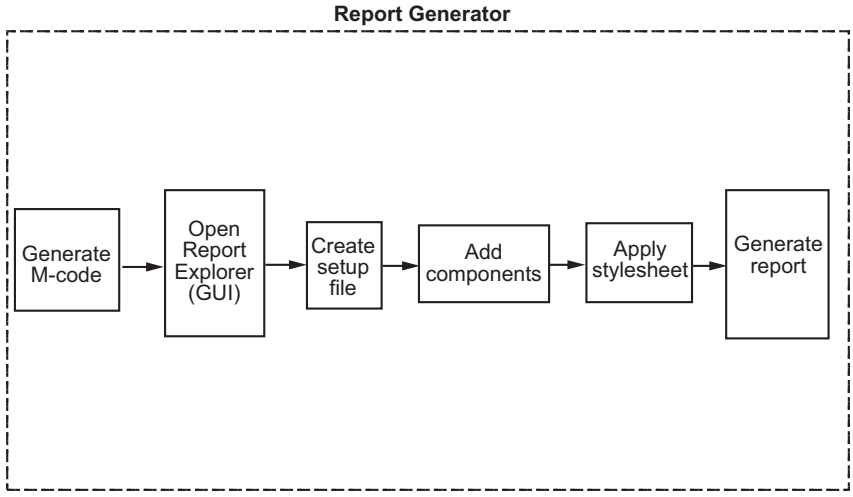
You can use the Report Explorer graphical interface to create reports without having to write code.

Using the programmatic approach, you can integrate report generation into analysis and testing applications. For more information, see “Programmatic Report Creation”.

Interactive Report Generation

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 “Working with Components”.
4. Apply styles and standards to the report by choosing an existing stylesheet or a custom stylesheet. For details on stylesheets and attributes, see “Layout Stylesheets”.
5. Generate the report. See “Generate Reports”.

The following figure illustrates a typical MATLAB Report Generator workflow.



To practice using this report creation workflow, see “Working with Components”.

How MATLAB Report Generator and MATLAB Software Interact

The MATLAB Report Generator and MATLAB software interact to create reports. You can access the Report Explorer from the MATLAB command line.

The following table summarizes these interactions.

User Interface	MATLAB Report Generator Interaction	Description
Report Explorer	The Report Explorer provides a graphical interface. For more information, see "Working with the Report Explorer" on page 1-8.	Use the Report Explorer to edit existing report templates, components, stylesheets, and attributes, or to customize your own.
MATLAB command line	Enter commands at the MATLAB command line to: • Start the Report Explorer • Create and modify report template files • Apply stylesheets • Specify output formats for reports • Generate reports	The following MATLAB commands work with the MATLAB Report Generator software: • `report` — Start the Report Explorer. • `setedit` — Edit a report template with the Report Explorer. • `rptconvert` — Convert a source file created by the report generation process to the desired output format. • `rptlist` — List `.rpt` files in the current path.

Report Components

Types of Report Components

Components are MATLAB objects that specify the content of a report. 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 can interactively combine components to create a report setup (see "Report Setup") that specifies the content of a particular report or type of report. You can then run the setup from the Report Explorer or the MATLAB command line to create instances of the report.

Use a combination of the following types of components in your report setup file, based on the goals for the report.

Type of Component	Description
"Report Structure Components" on page 5-4	Include a title page, chapters, sections, paragraphs, lists, tables, and other standard document structure elements.
"Table Formatting Components" on page 5-5	Organize generated content into tables.
"Property Table Components" on page 5-6	Display tables with property name/ property value pairs for objects.
"Summary Table Components" on page 5-17	Display tables with specified properties for objects.
"Logical and Looping Components" on page 5-21	Run child components a specified number of times. There are several looping components, including logical loops and Handle Graphics® loops.

Use the Report Explorer to add components to a report setup file and to specify component properties.

Working with the Report Explorer

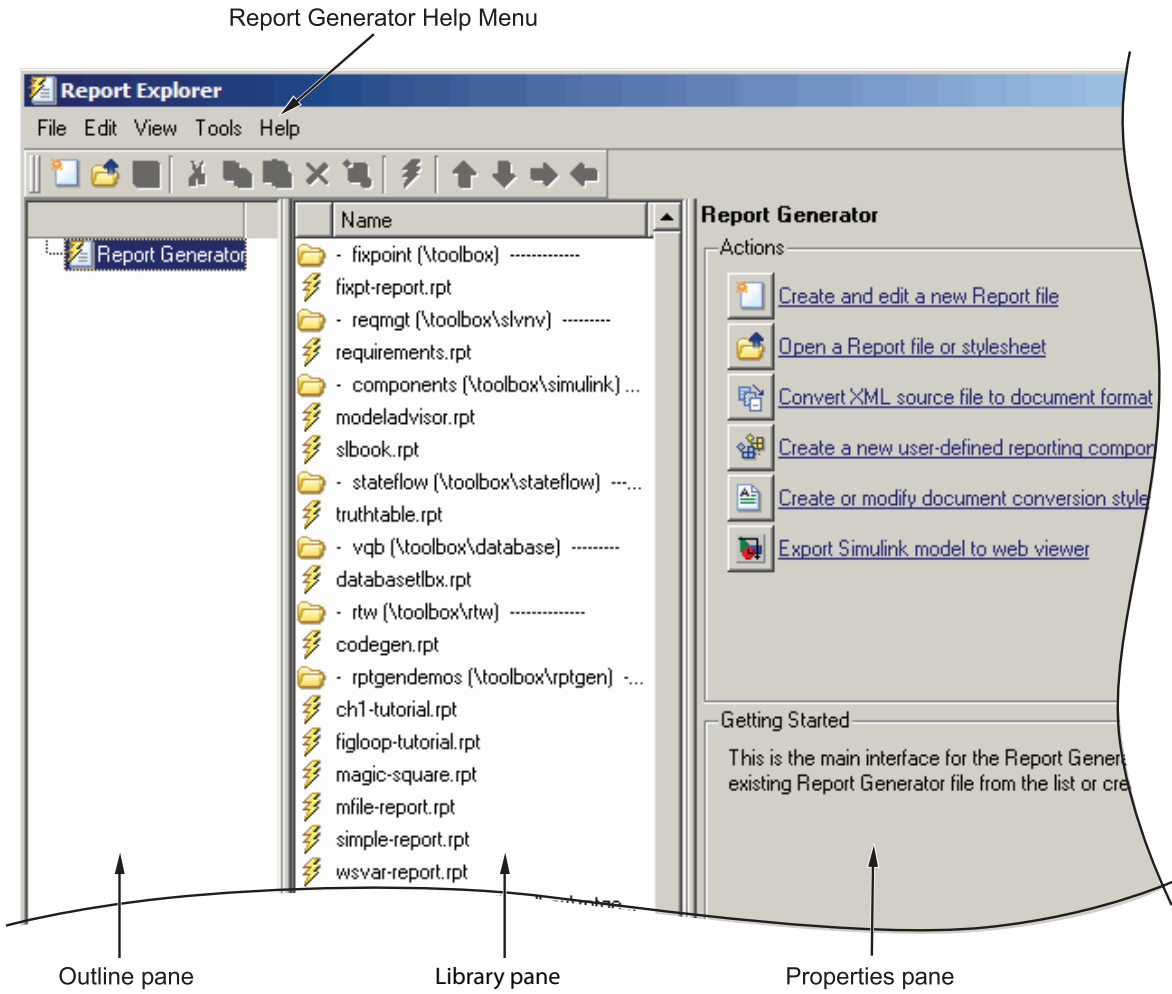
About the Report Explorer

Use the *Report Explorer* to:

- Create and modify report setup files.
- Apply stylesheets 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
Stylesheet is open.	Stylesheet 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
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.

Create Your First Report

- "Create a MATLAB Report" on page 2-2
- "Create a Report Setup File" on page 2-3
- "Add Report Content Using Components" on page 2-5
- "Generate a Report" on page 2-39

Create a MATLAB Report

This example shows how to create a report that 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 2-3
- "Add Report Content Using Components" on page 2-5

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 `web (HTML)` to create the report as an HTML file.
   c. In the **Report description** text box, replace the existing text with the following text.

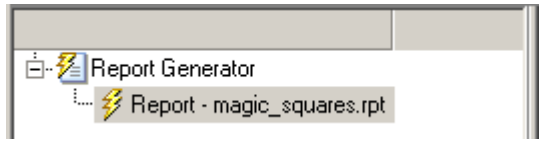
   Tip Copy and paste this text from the HTML documentation into the Report Explorer.

   ```
   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.



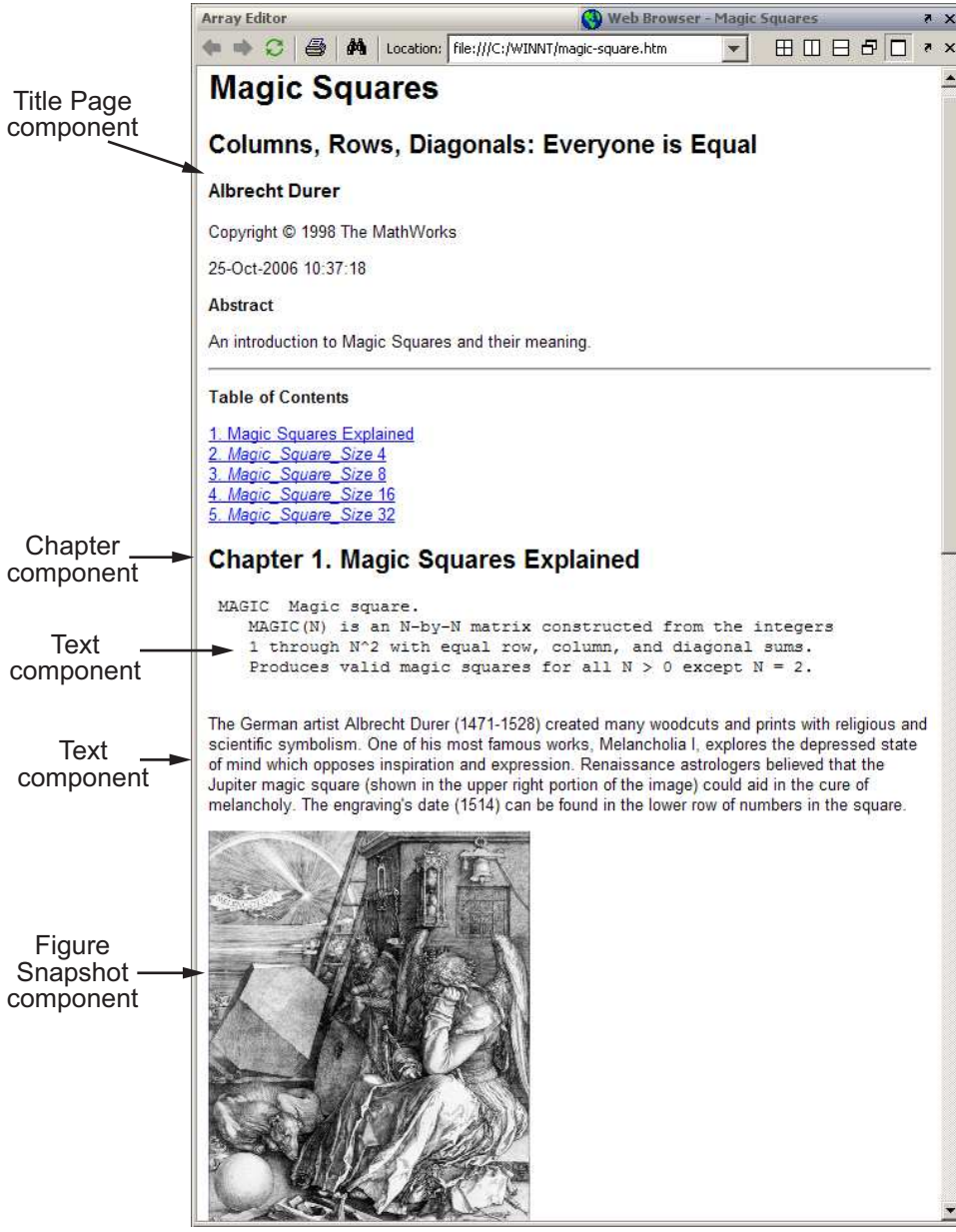
To create the content for the report, see “Add Report Content Using Components” on page 2-5.

Add Report Content Using Components

In this section...
“Report Components” on page 2-5
“Specify Report Variables” on page 2-7
“Create a Title Page” on page 2-9
“Add a Chapter” on page 2-12
“Add Introductory Text to the First Chapter” on page 2-14
“Add an Image” on page 2-16
“Create the Magic Squares and Their Images” on page 2-21
“Create a For Loop” on page 2-22
“Add a Chapter for Each Square” on page 2-24
“Determine the Matrix Size” on page 2-26
“Insert the Magic Square Size into the Report” on page 2-28
“Create the Magic Square” on page 2-29
“Add Display Logic” on page 2-32
“Display the Magic Square” on page 2-34

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.


Array Editor
Web Browser - Magic Squares
Location: file:///C:/WINNT/magic-square.htm
Title Page component
Chapter component
Text component
Text component
Figure Snapshot component
Magic Squares
Columns, Rows, Diagonals: Everyone is Equal
Albrecht Durer
Copyright © 1998 The MathWorks
25-Oct-2006 10:37:18
Abstract
An introduction to Magic Squares and their meaning.
Table of Contents
1. Magic Squares Explained
2. Magic_Square_Size 4
3. Magic_Square_Size 8
4. Magic_Square_Size 16
5. Magic_Square_Size 32
Chapter 1. Magic Squares Explained
MAGIC Magic square.
MAGIC(N) is an N-by-N matrix constructed from the integers
1 through N^2 with equal row, column, and diagonal sums.
Produces valid magic squares for all N > 0 except N = 2.
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.

Specify Report Variables

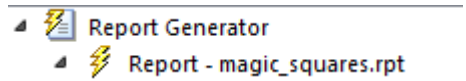
The magic squares report uses variables defined in the MATLAB workspace to specify the number and sizes of squares to display and whether to display the variables as tables of numbers or images of color-coded squares:

- 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

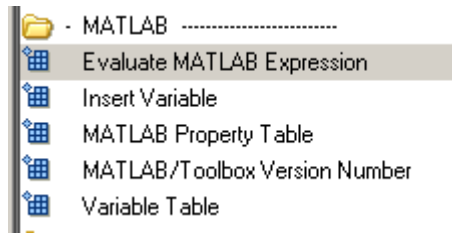
You could require that a user create these variables in the MATLAB workspace before running the report. However, a better solution is to let the report itself create the variables, using the Evaluate MATLAB Expression component.

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.



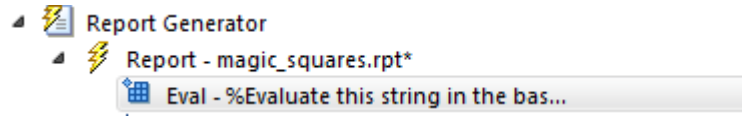
2 In the Library pane in the middle, under the `MATLAB` category, select `Evaluate MATLAB Expression`.



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.

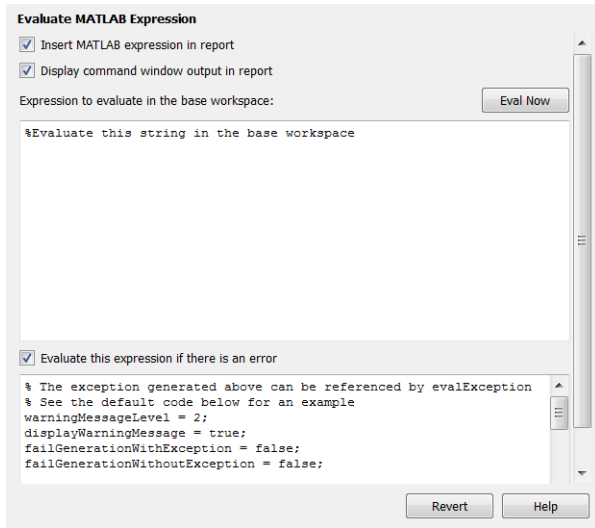
You cannot edit the component information in the Properties pane until you have added the component to the report.

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. Select **File** > **Save**.

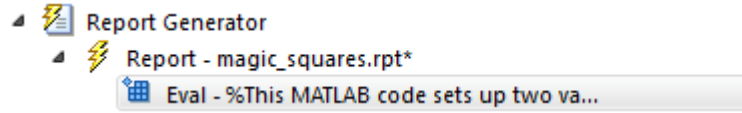
Create a Title Page

Note: This section builds on the previous tasks described in the step-by-step example summarized in "Create a MATLAB Report" on page 2-2.

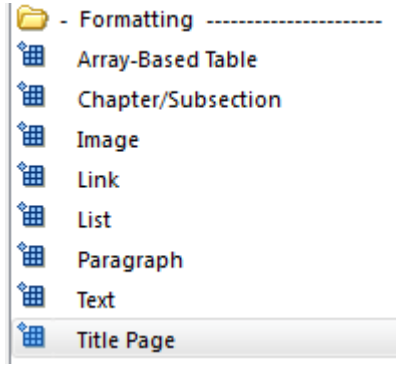
If you have not completed the previous sections of this example, see open the completed report setup file: `Magic Squares Report`.

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

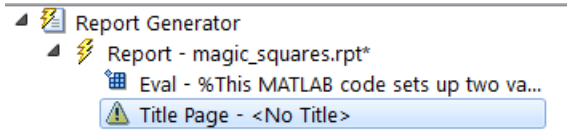
1 In the Outline pane on the left, select the `Eval` component.



2 In the Options pane in the middle, under the `Formatting` category, double-click `Title Page` to add the 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 also cannot have child components.



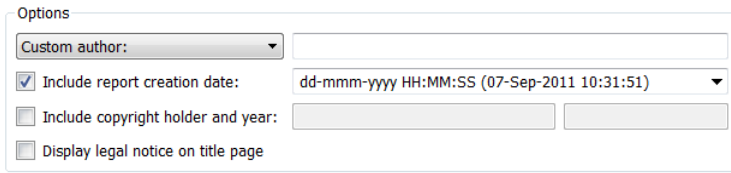
Note: To use a Title Page component, you need to have a Chapter component in your report. You have not yet added a Chapter component, so the Properties pane displays a message indicating that a chapters is required for the Title Page component to appear correctly. Because later in this example you add 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 following 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 selection list.



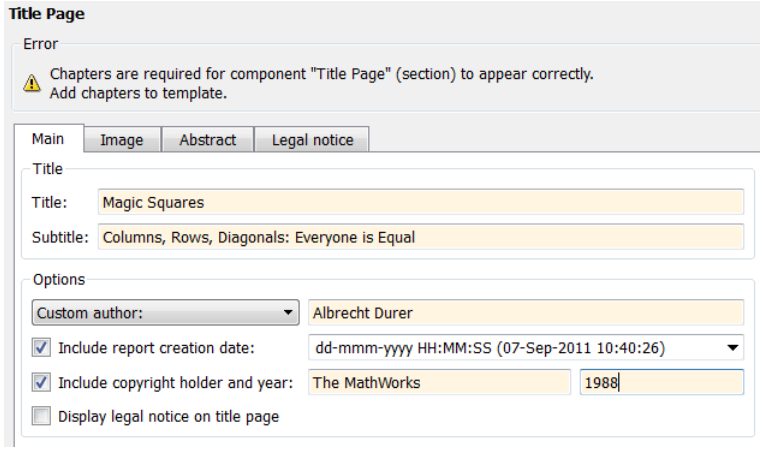
d In the field to the right of the **Custom author** field, enter `Albrecht Durer`.

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`.

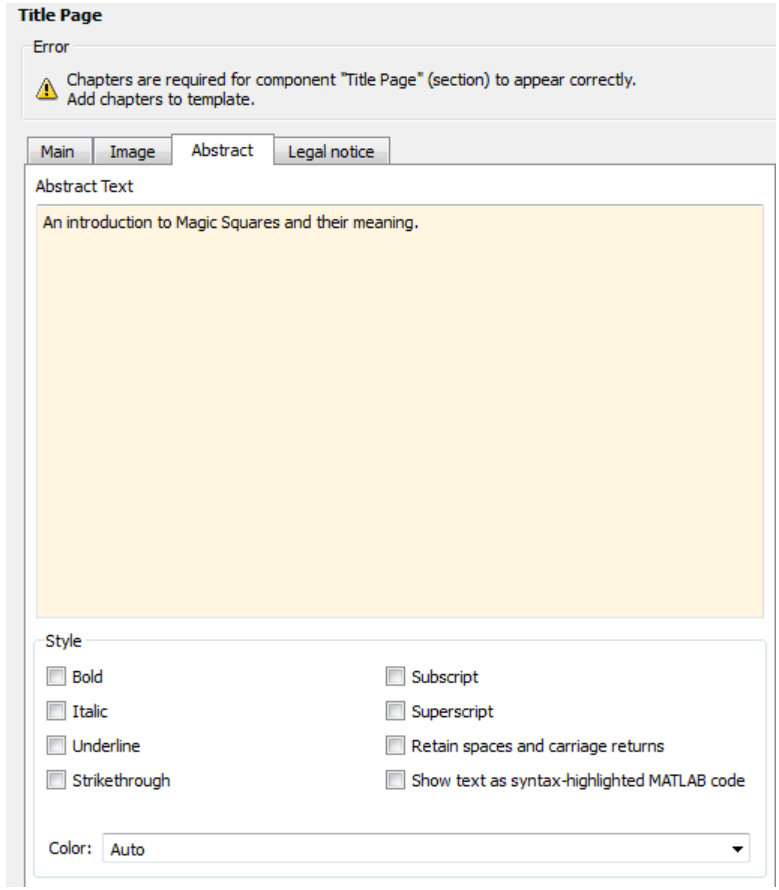
g In the second text box, enter `1988`.



4 In the Properties pane, click the **Abstract** tab and then enter the following text:

```
An introduction to Magic Squares and their meaning.
```

The pane should look as follows:



5 Save the report.

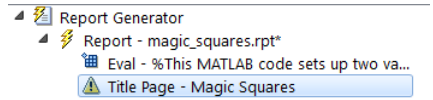
Add a Chapter

Note: This section builds on the previous tasks described in the step-by-step example summarized in “Create a MATLAB Report” on page 2-2.

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

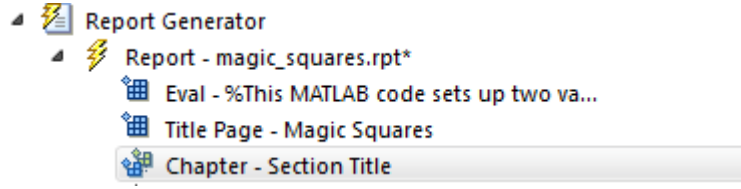
Add a chapter to the report by using the Chapter/Subsection component.

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



2. In the Library pane in the middle, under the `Formatting` category, double-click `Chapter/Subsection`.

   The Outline pane looks as follows.

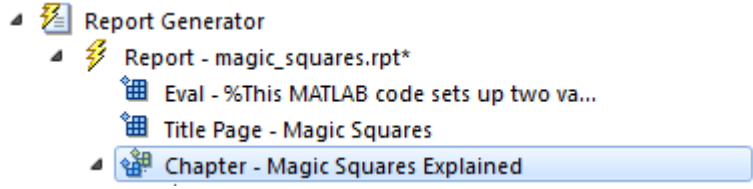


   The `Eval`, `Title Page`, and `Chapter` components are all child components of the report's top level, and are siblings of one another.

   The `Chapter` component can have child components. The next section explains how to add child components to this `Chapter` component.

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

   The Outline pane changes to reflect the chapter title.



4. Save the report.

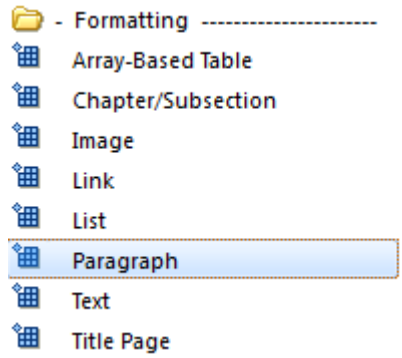
Add Introductory Text to the First Chapter

Note: This section builds on the previous tasks described in the step-by-step example summarized in "Create a MATLAB Report" on page 2-2.

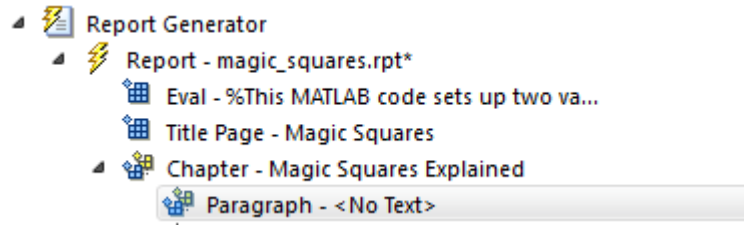
If you have not completed the previous sections of this example, see open the completed report setup file: `Magic Squares Report`.

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

1. In the Outline pane on the left, select the `Chapter` component.
2. In the Library pane in the middle, under the `Formatting` category, double-click `Paragraph`.



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



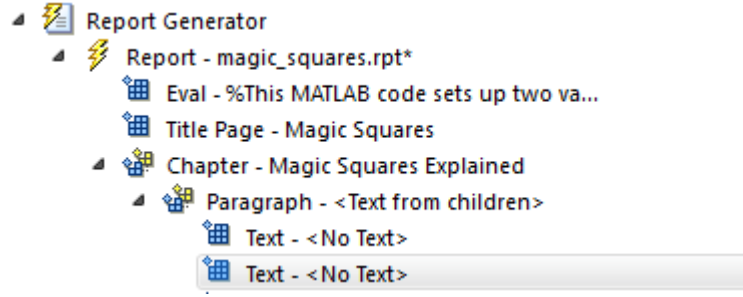
3 By default, the Paragraph component inherits its text from its child components. Add two `Text` components.

Note: The `Text` component must have the `Paragraph` component as its parent.

4 In the Library pane, under the `Formatting` category, double-click `Text`.

5 Double-click `Text` again to add a second component.

The Outline pane looks as follows.



6 In the Outline pane, select the first `Text` component.

7 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.

8 In the Outline pane, select the second `Text` component.

9 In the **Text to include in report** text box, enter the following text.

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

```
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.
```

10 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 "Create a MATLAB Report" on page 2-2.

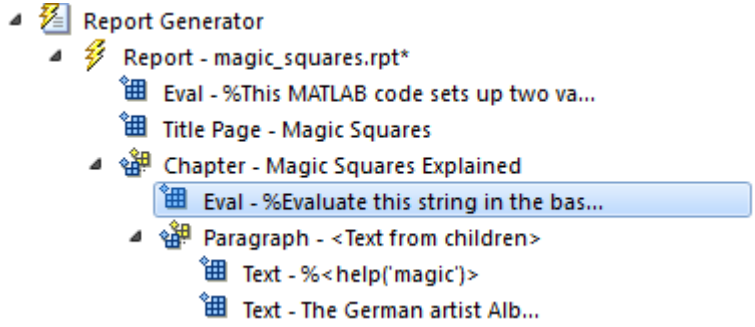
If you have not completed the previous sections of this example, see open the completed report setup file: `Magic Squares Report`.

The next steps create an image of Albrecht Dürer and include it in the report.

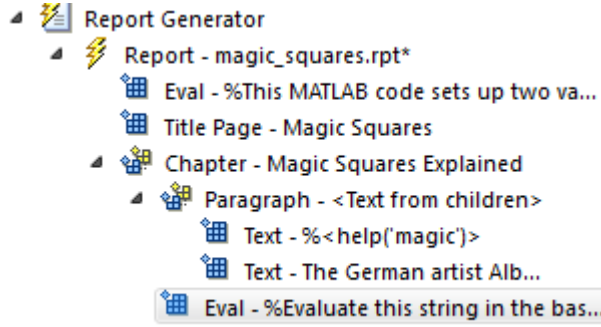
1 In the Outline pane on the left, select the `Chapter` component.

2 In the Library pane in the middle, under the MATLAB category, double-click `Evaluate MATLAB Expression`.

The new component becomes a child of the `Chapter` component.



3 Move the `Eval` component under the `Paragraph` component so that the image follows the introductory text by clicking the **down** arrow on the toolbar.



4 With the `Eval` component still selected, do the following in the Properties pane on the right:

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 the following MATLAB code.

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

```
%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
```

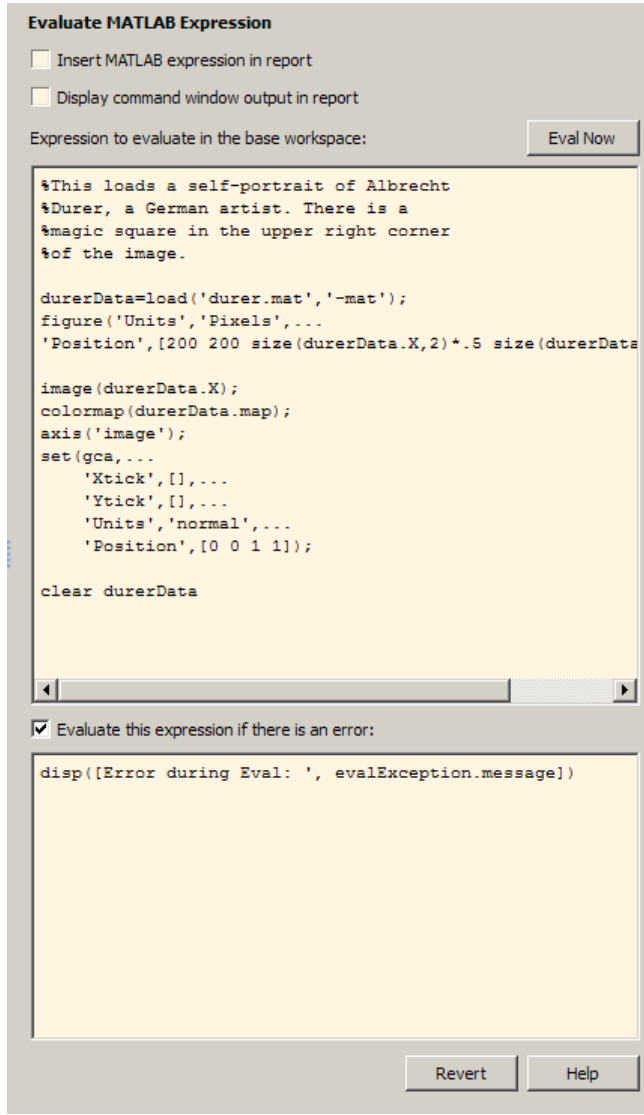
This MATLAB code displays the Dürer etching in a MATLAB figure window.

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

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

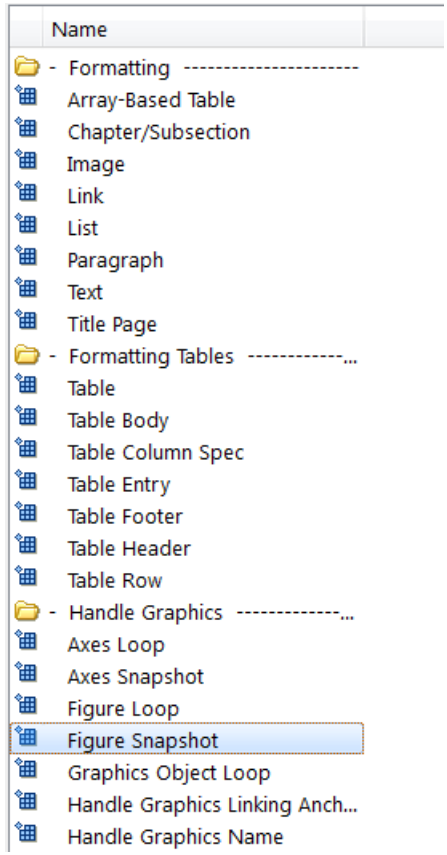
This code executes if an error occurs while loading the Dürer etching.

The Properties pane on the right looks as follows.



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

6 In the Library pane in the middle, under the `Handle Graphics` category, double-click `Figure Snapshot`.



To inline an image component (such as Image or Figure Snapshot), include it within 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 on-screen 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 the `MATLAB` category, double-click `Evaluate MATLAB Expression`.

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 the following text:

```
%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 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

In the next steps, you 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 essentially repetitive task. To create the magic squares and their images, you perform these tasks:

- "Create a For Loop" on page 2-22
- "Add a Chapter for Each Square" on page 2-24
- "Determine the Matrix Size" on page 2-26
- "Insert the Magic Square Size into the Report" on page 2-28

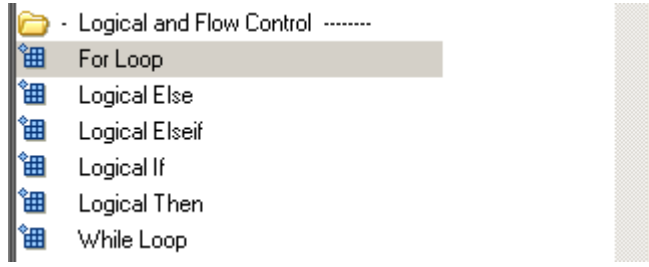
- "Create the Magic Square" on page 2-29
- "Add Display Logic" on page 2-32
- "Display the Magic Square" on page 2-34

Create a For Loop

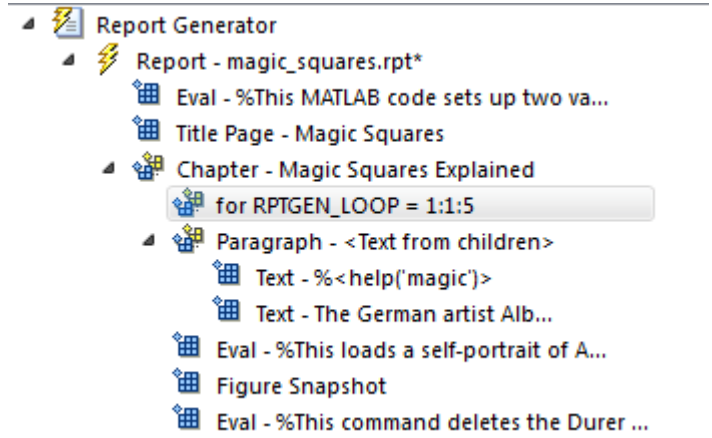
Note: This section builds on the previous tasks described in the step-by-step example summarized in "Create a MATLAB Report" on page 2-2.

If you have not completed the previous sections of this example, see 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 the `Logical and Flow Control` category, double-click `For Loop`.



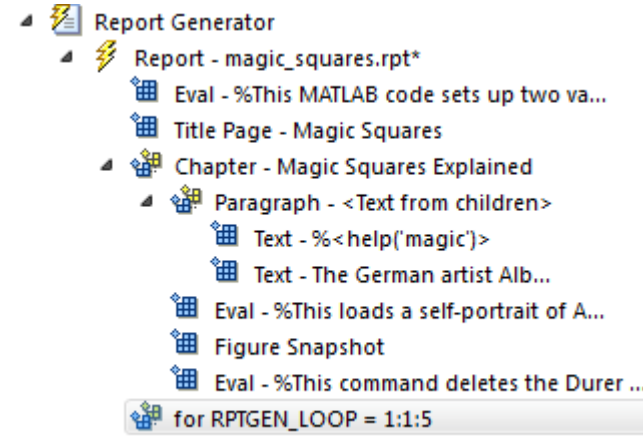
The Outline pane looks as follows.



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



5 In the Properties pane on the right:

a In the **End** text box, replace the existing text with the following text:

```
length(magicSizeVector)
```

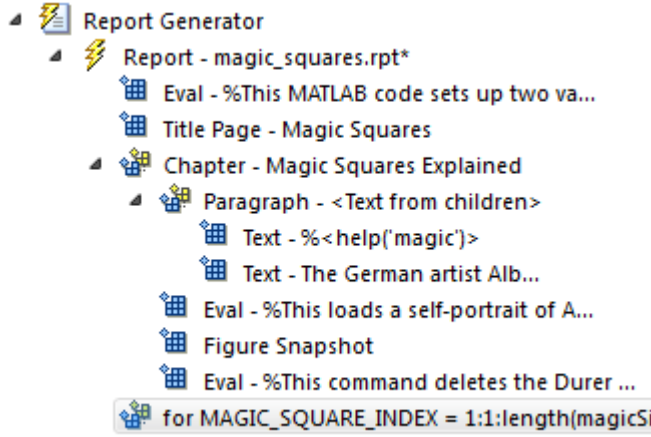
This is the length of the vector that contains the various sizes for the magic square matrices.

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.

The Outline pane looks as follows.



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 "Create a MATLAB Report" on page 2-2.

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

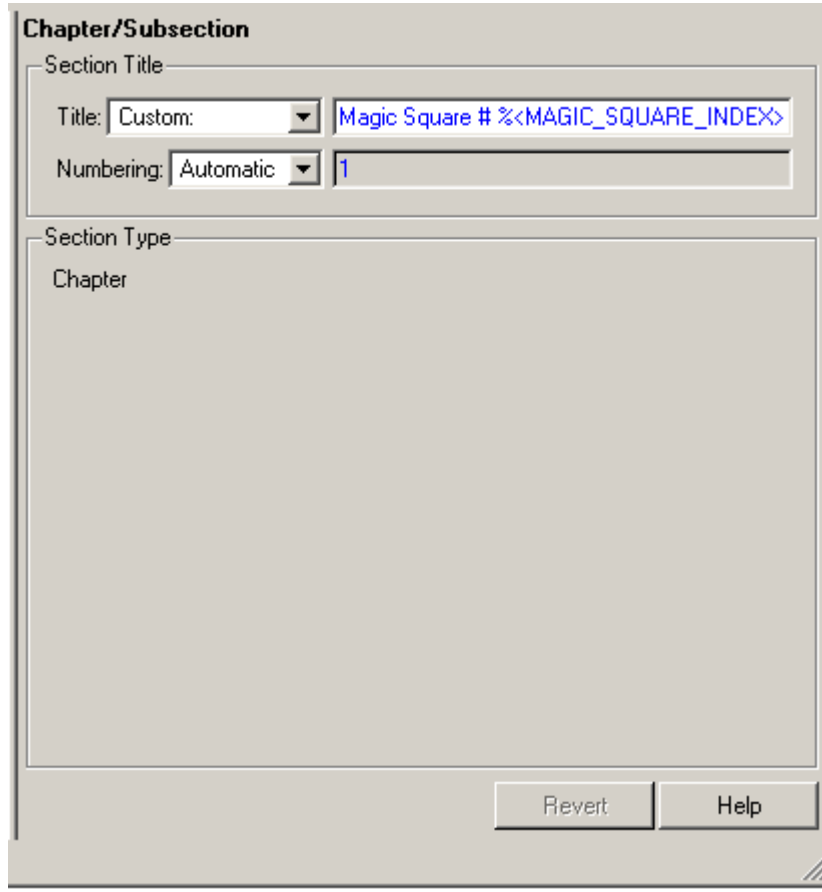
Next create a chapter for each square by adding a Chapter component to the report as a child of the For Loop component. This causes the Report Generator to create a chapter on each iteration of the For Loop during report generation.

1. In the Outline pane on the left, select the `for` component.
2. In the Library pane in the middle, under the `Formatting` category, double-click `Chapter/Subsection`.

   It becomes a child of the `for` component.
3. In the Properties pane on the right, select `Custom` from the **Title** list and enter the following for the chapter title:

   ```
   Magic Square # %<MAGIC_SQUARE_INDEX>
   ```

   The Properties pane looks as follows.



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 "Create a MATLAB Report" on page 2-2.

If you have not completed the previous sections of this example, see 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 the `MATLAB` category, double-click `Evaluate MATLAB Expression`.
3. 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 the following text:

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

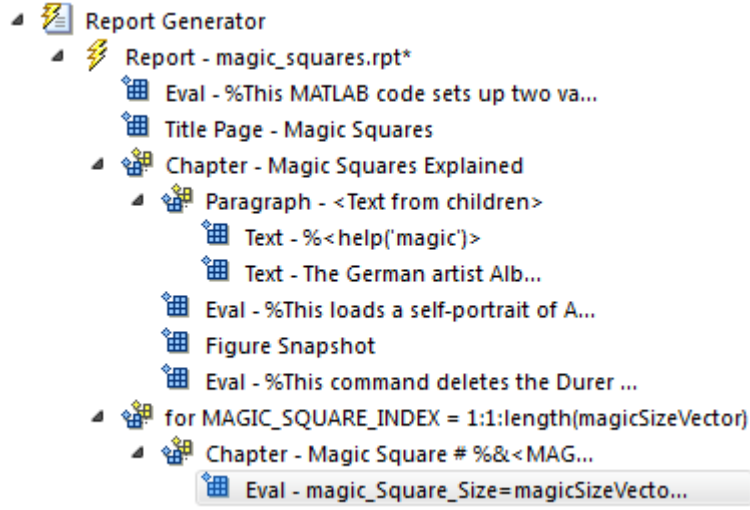
   This command extracts the next size for the magic square from the vector of sizes initialized in the first `Eval` component of the report. The variable `magic_Square_Size` represents the size of the current magic square being processed.

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

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

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

   The Outline pane looks as follows.



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 "Create a MATLAB Report" on page 2-2.

If you have not completed the previous sections of this example, see 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 the `Formatting` category, double-click `Paragraph`.

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).

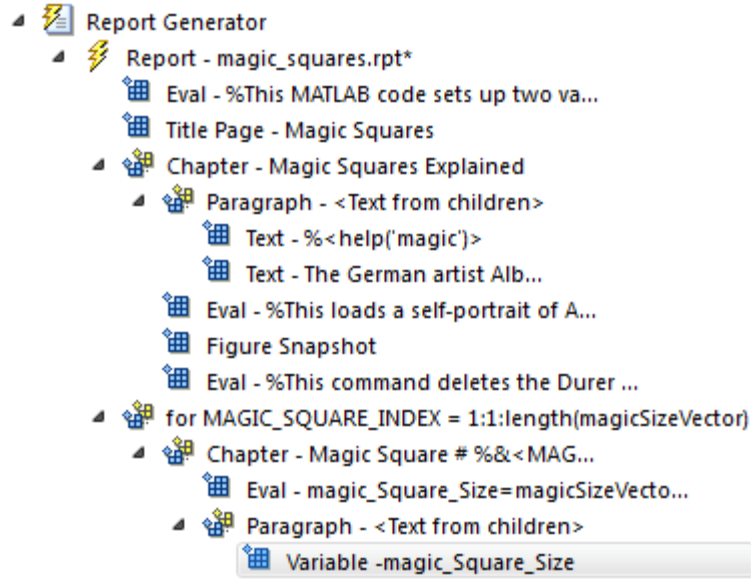
4 In the Library pane, under the `MATLAB` category, double-click `Insert Variable`.

5 In the Properties pane on the right:

   a In the **Variable name** text box, enter `magic_Square_Size`.

   b In the **Display as** list, select `Inline text`.

   The Outline pane looks as follows.



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 “Create a MATLAB Report” on page 2-2.

If you have not completed the previous sections of this example, see 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 the `MATLAB` category, double-click `Evaluate MATLAB Expression`.

   Make this component a sibling of the `Paragraph` component, not a child, as described in the next two steps.
3. In the Outline pane, select the `Eval` component.
4. Click the left arrow on the toolbar to make the `Eval` component a sibling of the previous `Paragraph` component.
5. 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 the following MATLAB code.

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

```
%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;
```

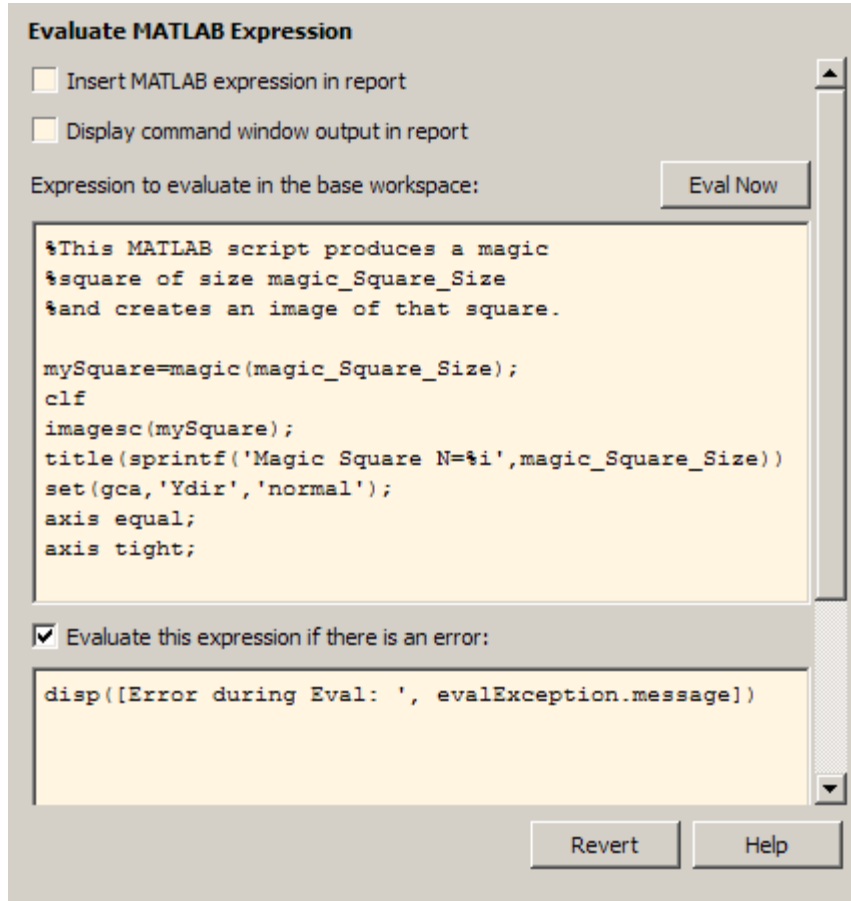
   This code creates a magic square matrix `mySquare` of size `magic_Square_Size`, and opens an image of that matrix in the MATLAB figure window.

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

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

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

The Properties pane looks as follows.



6 Save the report.

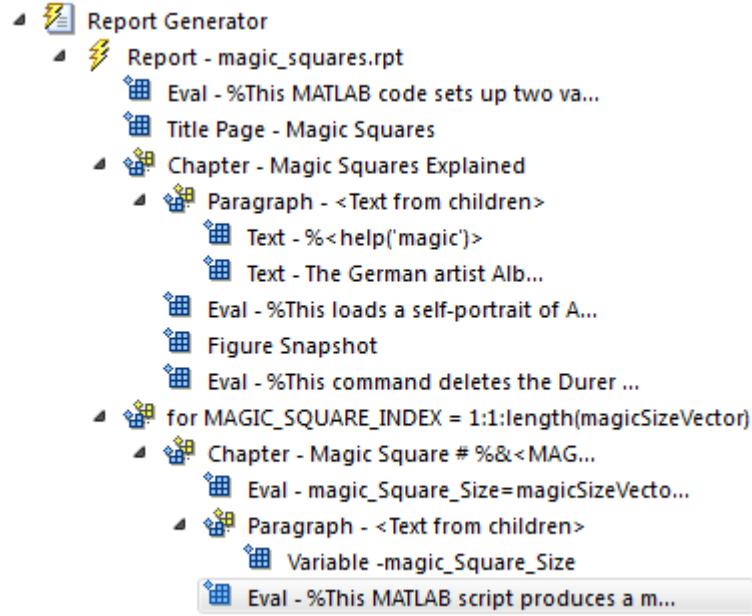
Add Display Logic

Note: This section builds on the previous tasks described in the step-by-step example summarized in "Create a MATLAB Report" on page 2-2.

If you have not completed the previous sections of this example, see 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 the `Logical and Flow Control` category, double-click `Logical If`.

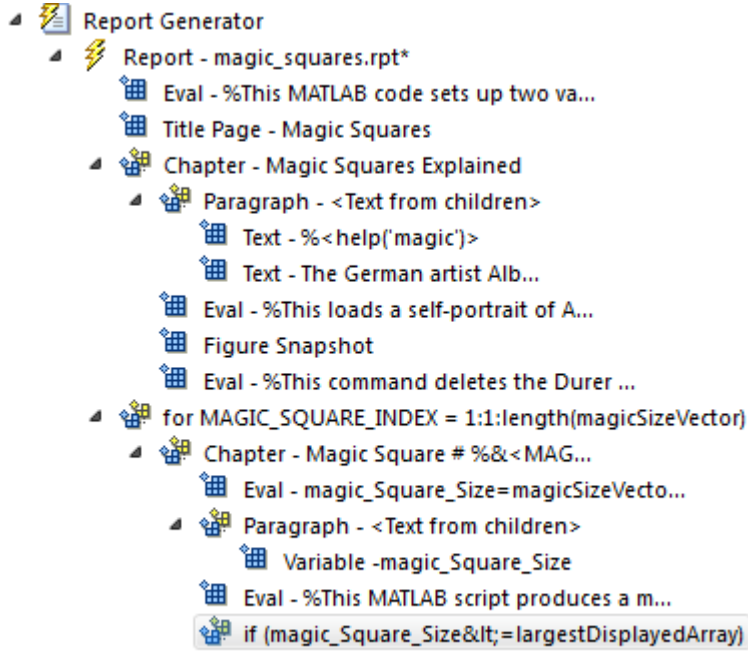
3 On the Properties pane on the right, in the **Test Expression** text box, replace the existing text with the following text:

```
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.

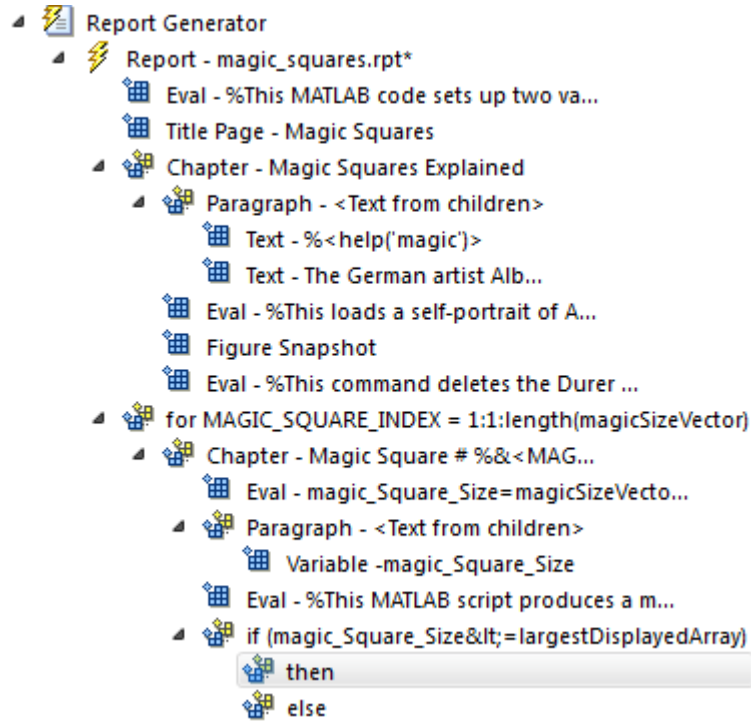


5 On the Library pane, under `Logical and Flow Control`, double-click `Logical Else`.

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

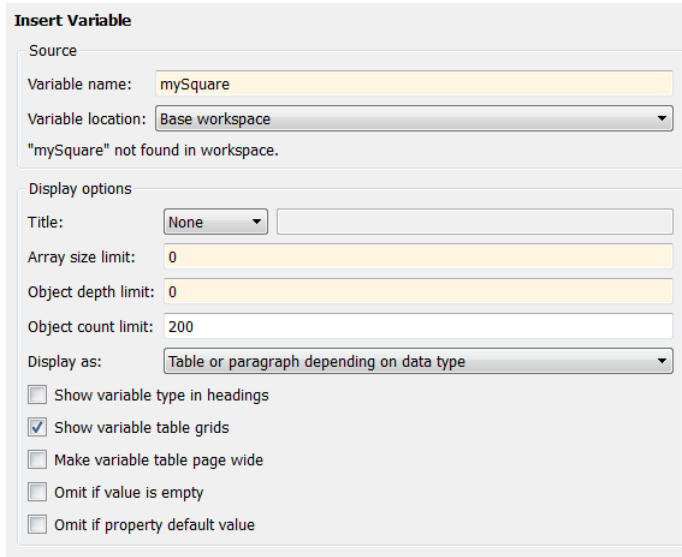
Note: This section builds on the step-by-step example presented in “Create a MATLAB Report” on page 2-2.

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 the `MATLAB` category, 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. In the **Title** list, select `None`.
   c. In the **Array size limit** text box, enter `0`.

   The Properties pane on the right looks as follows.



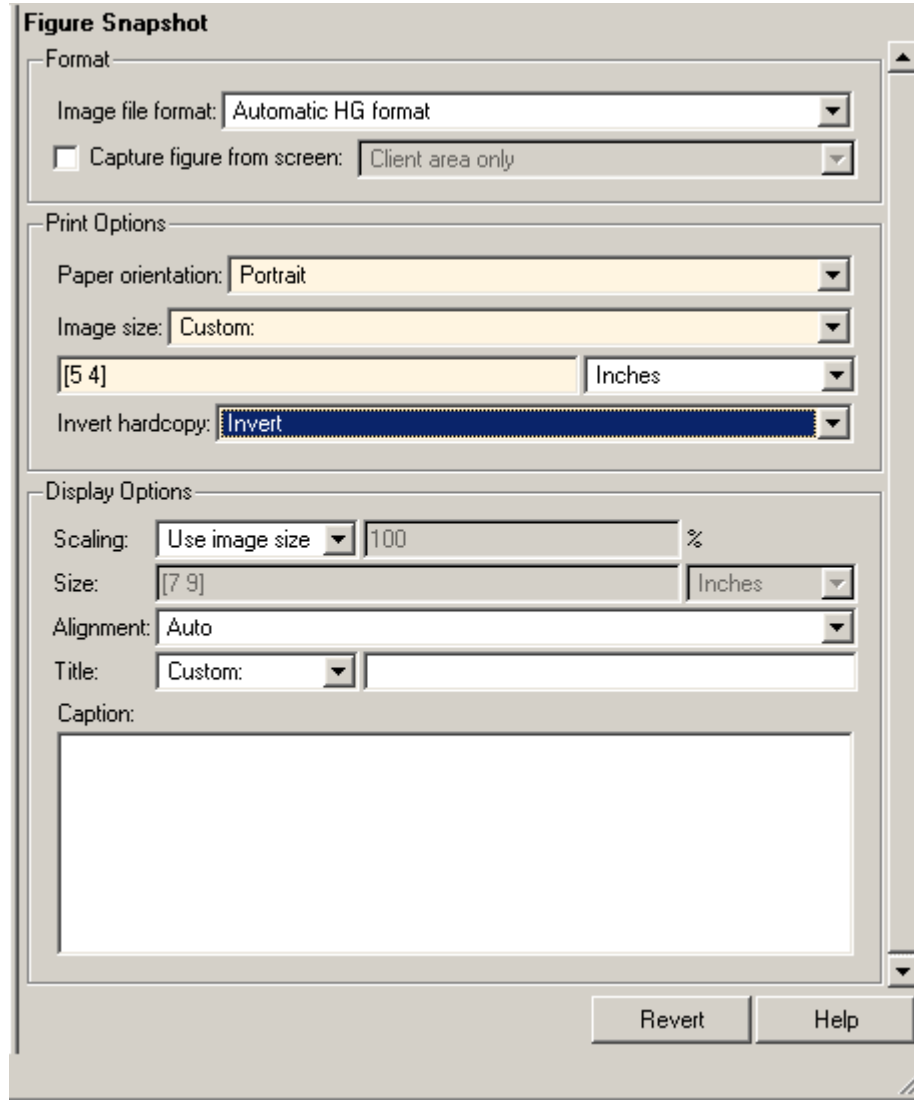
   This Variable component displays the magic square matrix, stored in the `mySquare` variable.
4. In the Outline pane, select the `else` component.
5. In the Library pane, under the `Handle Graphics` category, double-click `Figure Loop`.

   Do not change its properties.
6. In the Outline pane, select the `Figure Loop` component.

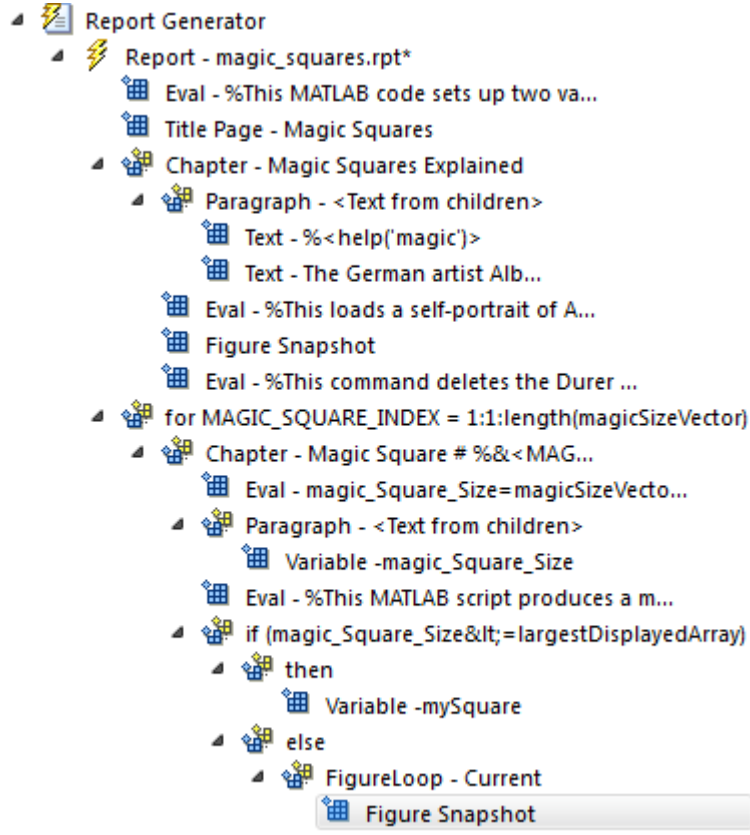
7. In the Library pane, under the `Handle Graphics` category, double-click `Figure Snapshot`.
8. In the Properties pane:
   a. In the **Paper orientation** list, select `Portrait`.
   b. In the **Image size** list, select `Custom`.
   c. Under the **Image size** list, enter `[5 4]` for the custom image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

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

      The Properties pane on the right looks as follows.



The Outline pane looks like the following.



9 Save the report.

Generate a Report

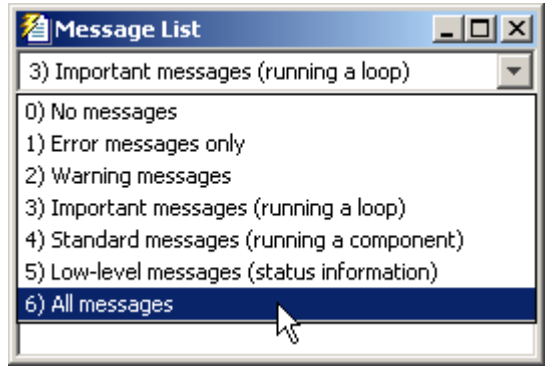
Note: This section builds on the step-by-step example presented in "Create a MATLAB Report" on page 2-2.

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

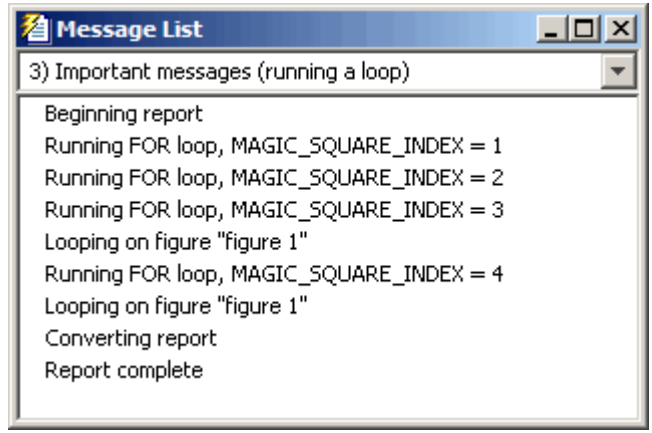
Now the report contains all the components it needs.

On the toolbar, click the Report icon to generate the report. The following dialog box appears.

1 A Message List window appears, displaying informational and error messages as the report processes. While the report generates, specify the level of detail you would like the Message List window to display. Options range from 0 (least detail) to 6 (most detail). Click the list located under the title bar of the Message List window to choose an option, as shown here.

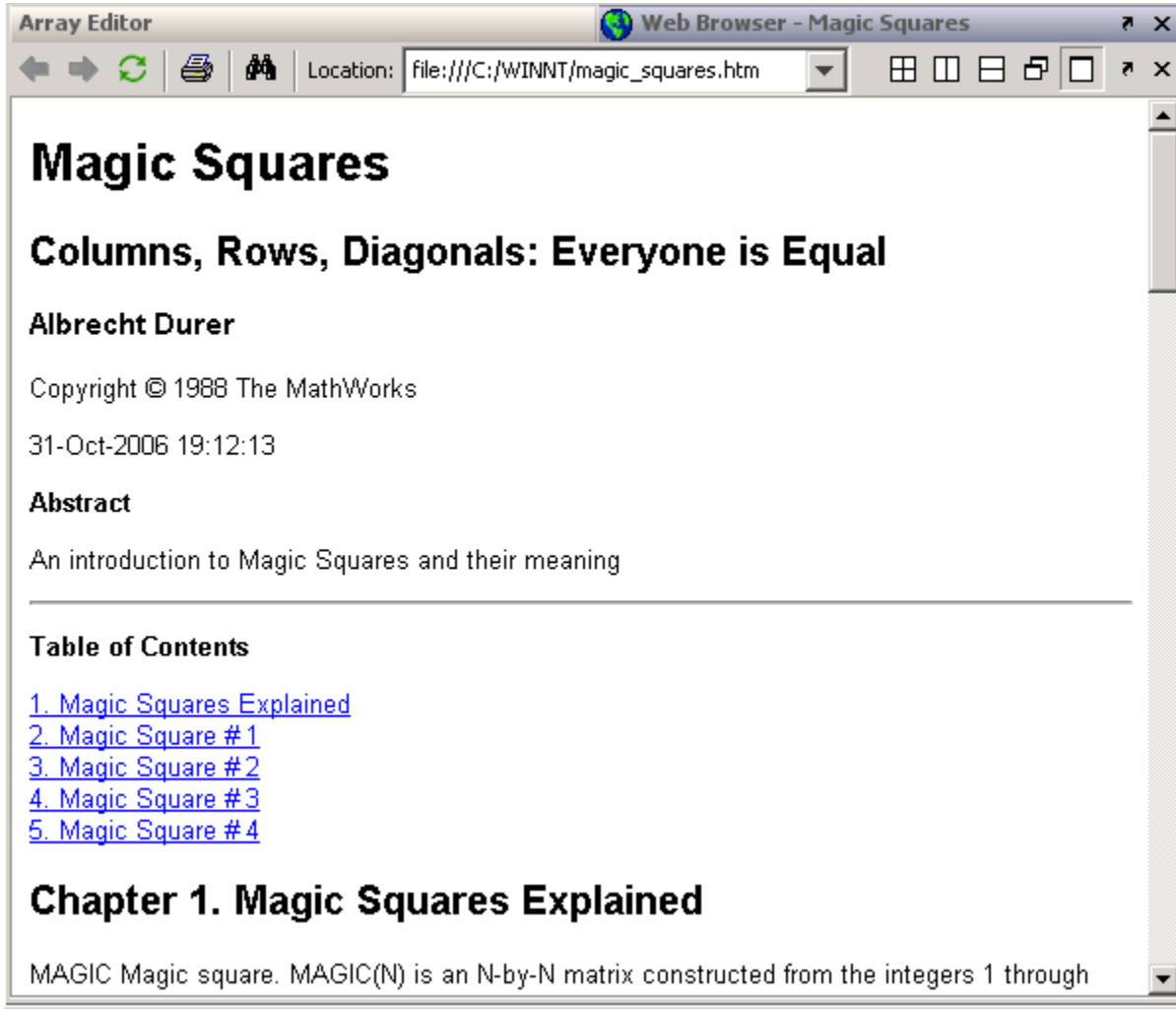


Message level 3 (Important messages) is used for the remainder of this example.



2. An image of the etching appears briefly.
3. Images of two magic square images of sizes 16 and 32 appear briefly.
4. In the Outline pane on the left of your Report Explorer window, each component of the report setup file is highlighted as it is executed.

At the beginning of this example you specified HTML as the output format of this report. When processing finishes, the MATLAB Web browser opens and displays the report's HTML file.


Array Editor
Web Browser - Magic Squares
Location:
file:///C:/WINNT/magic_squares.htm
Magic Squares
Columns, Rows, Diagonals: Everyone is Equal
Albrecht Durer
Copyright © 1988 The MathWorks
31-Oct-2006 19:12:13
Abstract
An introduction to Magic Squares and their meaning
Table of Contents
1. Magic Squares Explained
2. Magic Square #1
3. Magic Square #2
4. Magic Square #3
5. Magic Square #4
Chapter 1. Magic Squares Explained
MAGIC Magic square. MAGIC(N) is an N-by-N matrix constructed from the integers 1 through

Set Up a Report

- "Report Setups" on page 3-2
- "Create a New Setup File" on page 3-4
- "Open a Report Setup" on page 3-6
- "Close a Report Setup" on page 3-8
- "Save a Report Setup" on page 3-9
- "Load Report Setup into MATLAB Workspace" on page 3-10
- "Insert Components" on page 3-11
- "Set Component Properties" on page 3-12
- "Move Components" on page 3-13
- "Delete Components" on page 3-15
- "Deactivate Components" on page 3-16
- "Send Components to the MATLAB Workspace" on page 3-17

Report Setups

In this section...
“Setup Hierarchy” on page 3-2
“Setup Files” on page 3-2
“Create a Report Setup” on page 3-3

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

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

Once you create 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.
- Insert components to define the content and format of the report.
- Set component properties.
- Save the setup.

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 3-4
"Create Setup File Programmatically" on page 3-4
"Working with Setup Files" on page 3-4

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 3-6
- "Close a Report Setup" on page 3-8

- “Save a Report Setup” on page 3-9

Open a Report Setup

In this section...
"Opening a Setup on the MATLAB Path" on page 3-6
"Opening a Setup Not on the MATLAB Path" on page 3-7
"Opening a Setup Programmatically" on page 3-7

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

In this section...
“Close a Setup Using the Report Explorer” on page 3-8 “Close a Setup Programmatically” on page 3-8

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

In this section...
“Save a Setup Under Its Existing Name” on page 3-9
“Save a Setup Under a New Name” on page 3-9

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-fop';
setupRoot.execute;
```

Insert Components

In this section...
“Point-and-Click Method” on page 3-11
“Drag-and-Drop Method” on page 3-11
“Fix Context Violations” on page 3-11

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 3-13).

Set Component Properties

In this section...
“Edit Component Property Values” on page 3-12 “Computed Property Values” on page 3-12

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 string property of a component. A string property is a property whose value is a string of text. To specify a MATLAB expression as a string property value, in the Properties dialog box, in the property edit box, enter `%<expr>`, where `expr` is a MATLAB expression that evaluates to a string.

Move Components

In this section...
"Point-and-Click Method" on page 3-13 "Drag-and-Drop Method" on page 3-14

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
Move Up	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.
Move Down	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.
Move Left	Moves a component up one level in the hierarchy. The component becomes a sibling of its former parent.

Move Command or Button	Effect
Move Right	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 4-2
- "Select Report Generation Options" on page 4-4
- "Report Generation Preferences" on page 4-13
- "Change Report Locale" on page 4-17
- "Convert XML Documents to Different File Formats" on page 4-18
- "Create a Report Log File" on page 4-21
- "Generate MATLAB Code from Report Setup File" on page 4-22
- "Troubleshooting Report Generation Issues" on page 4-25

Generate a Report

In this section...
“Run a Report” on page 4-2
“Report Output Options” on page 4-2

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)
- Stylesheet 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 4-4
- “Location of Report Output File” on page 4-10
- “Create a Report Log File” on page 4-21

- "Report Description" on page 4-12
- "Change Report Locale" on page 4-17

Select Report Generation Options

In this section...
“Report Options Dialog Box” on page 4-4
“Report Output Format” on page 4-4
“PDF Stylesheets” on page 4-7
“Web Stylesheets” on page 4-7
“RTF (DSSSL Print) and Word Stylesheets” on page 4-8
“Report Generation Processing” on page 4-8
“Location of Report Output File” on page 4-10
“Report Description” on page 4-12

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 4-13.

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 stylesheets.

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)`

- `PDF (from Word template)`
- `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 6-4. For information on customizing templates, see "Customize Report Conversion Templates".

File Format Using Report Explorer Stylesheets

The output formats that do not include `from template` in the name use Report Explorer stylesheets for formatting. For those output formats, select a stylesheet from the **Stylesheet** list.

Viewer	Format	Description	Stylesheets
Adobe® Acrobat® Reader	Adobe Acrobat (PDF)	Produce a PDF that you can view using Adobe Acrobat Reader software. See "PDF: Image Formats" on page 4-6.	PDF (see "PDF Stylesheets" on page 4-7)
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 4-6.	Print (see "RTF (DSSSL Print) and Word Stylesheets" on page 4-8)
DocBook	`DocBook (no transform)`	Produce a report in DocBook format	N/A

Tip To create and use customized styles, see "Create a New Stylesheet" on page 8-4.

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.

Stylesheets

For each output format, you can choose from several stylesheets for each report output format. For details, see:

- “PDF Stylesheets” on page 4-7
- “Web Stylesheets” on page 4-7
- “RTF (DSSSL Print) and Word Stylesheets” on page 4-8

Note: Some Web and Print stylesheets include a generated list of titles, which includes table titles and figures with titles.

PDF Stylesheets

PDF Stylesheet	Description
`Default print stylesheet`	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 Stylesheets

Web Stylesheet	Description
`Default HTML stylesheet`	HTML on a single page
`Simulink book HTML stylesheet`	HTML on multiple pages; suppresses chapter headings and table of contents
`Truth Table HTML stylesheet`	HTML on multiple pages; suppresses chapter headings and table of contents
`Multi-page Web`	HTML, with each chapter on a separate page

Web Stylesheet	Description
`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
`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 Stylesheets

RTF or Word Stylesheet	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 several options for controlling report processing.

Option	Purpose
View report after generation	When report generation finishes, the viewer associated with the report output format displays the report. **Note:** On Linux and Macintosh platforms, the report output displays in Apache OpenOffice™, which must be installed in `/Applications/OpenOffice.app`. To view the report manually, open the file from the location specified in the **Report Options** for the report, under **Report File Location**.
Auto save before generation	Automatically save the report setup file before you generate a report.

Option	Purpose
Compile model to report on compiled information	Ensure that a report reflects compiled values. 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. 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. **Note:** When you select this option, whenever report generation requires simulating the model (for example, the report includes a Model Simulation 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. To minimize compilations, consider using separate reports to report on the contents of a model and on the results of simulating that model.
Evaluate this string after generation	Specify MATLAB code for processing to occur after the report is generated. For example, you could specify to close a model.

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 **Browse** button (...) to select from a list of directories.

You can use `%<VariableName>` notation to specify a folder in the **Custom** text box. For more information, see "%<VariableName> Notation" on page 10-102 on the `Text` component reference page.

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. For more information, see "%<VariableName> Notation" on page 10-102 on the `Text` component reference page.

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 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.

Report Generation Preferences

In this section...
“Report Generator Preferences Pane” on page 4-13
“File Format and Extension” on page 4-14
“Image Formats” on page 4-15
“Report Viewing” on page 4-15
“Reset to Defaults” on page 4-16
“Edit HTML Command” on page 4-16

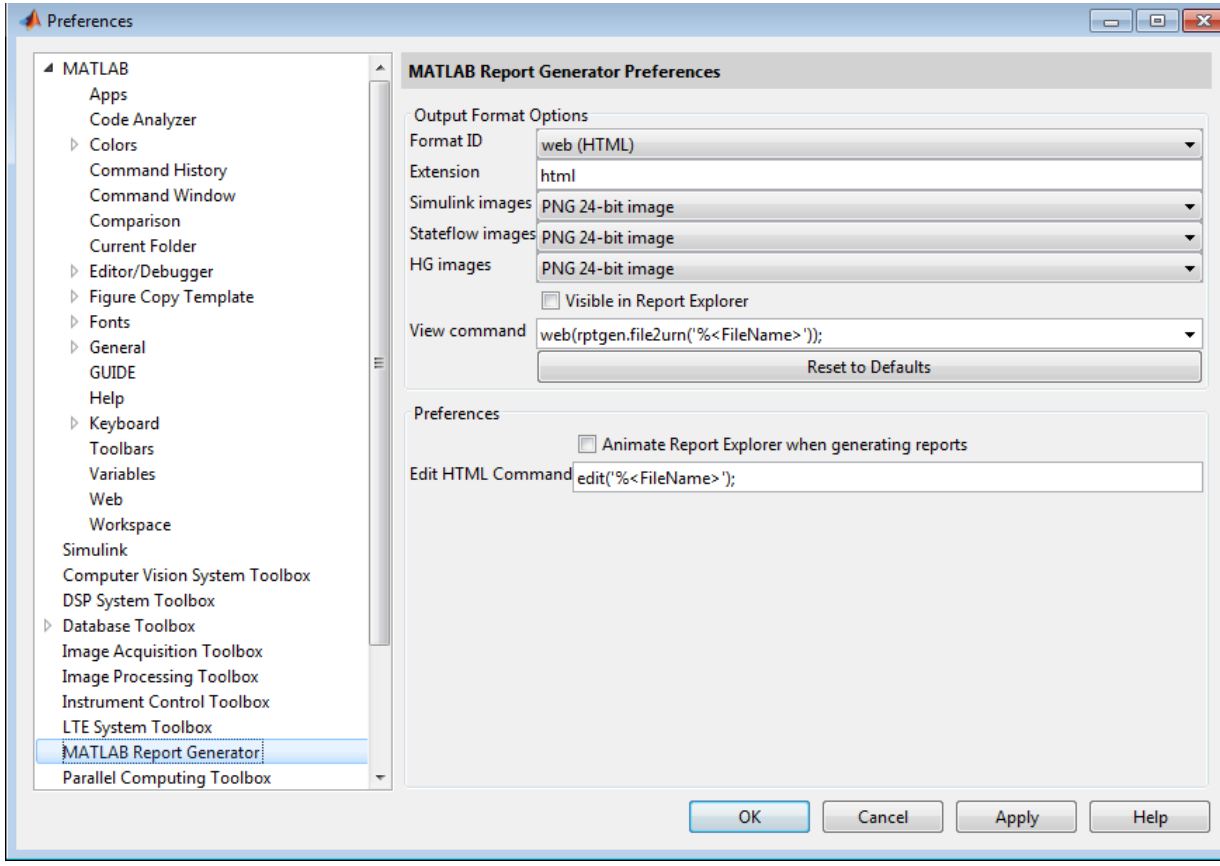
Report Generator Preferences Pane

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 4-4.

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, Microsoft Word, or LaTex.

Note: For reports that use the Word Document format, you must have Microsoft Word installed on the machine that you use to generate the report.

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
`Simulink Images`	Specify the format for Simulink images to include in the report.
`Stateflow Images`	Specify the format for Stateflow charts to include in the report.
`HG Images`	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
`View command`	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).

Preference	Purpose
`Animate Report Explorer when generating reports`	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 Conversion Templates” on page 6-2). 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 string 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 4-18
"Convert XML Documents Using the Report Explorer" on page 4-18
"Convert XML Documents Using the Command Line" on page 4-20
"Edit XML Source Files" on page 4-20

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`.

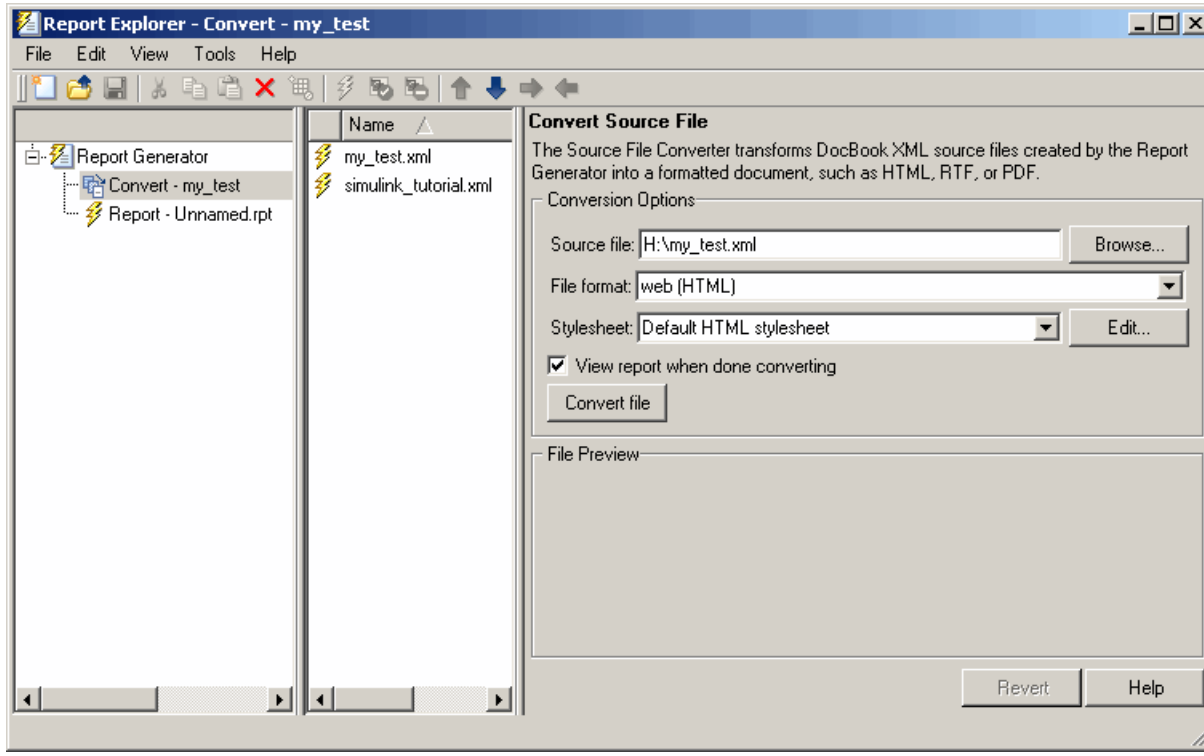
Note: The report-generation process can only convert XML source files created by the latest version of the software.

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 stylesheet:

a In the **File format** text box, select an output format.

b In the **Stylesheet** text box, select a stylesheet. The stylesheet choice depends on the specified output format. You can use a predefined or customized stylesheet.

For more information about available formats and predefined stylesheets, see “Report Output Format” on page 4-4.

For more information about customizing stylesheets, see “Create a New Stylesheet” on page 8-4.

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 `rptconvert`function.

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.

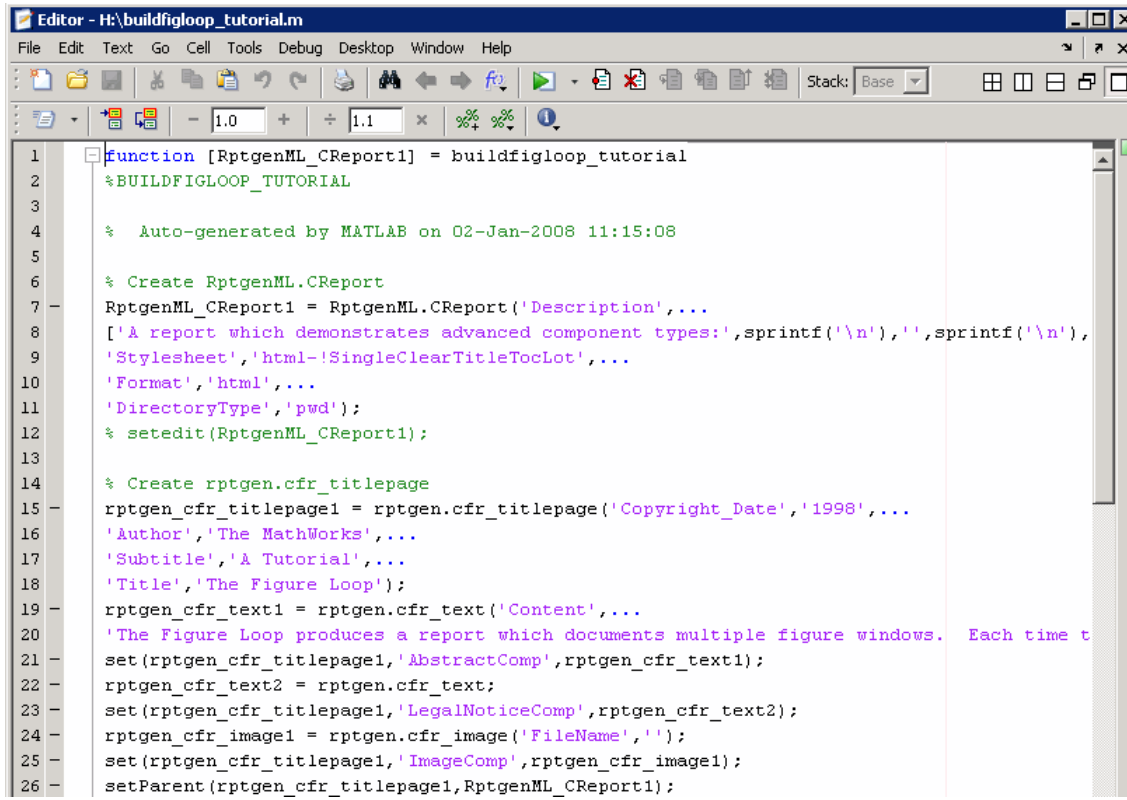
To generate a MATLAB file, load a report setup file into the Report Explorer and click **File > Generate MATLAB File**. After the MATLAB file generates, it opens in the MATLAB Editor. The filename for the generated file is the file name of the report setup file , preceded by "build."

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**.

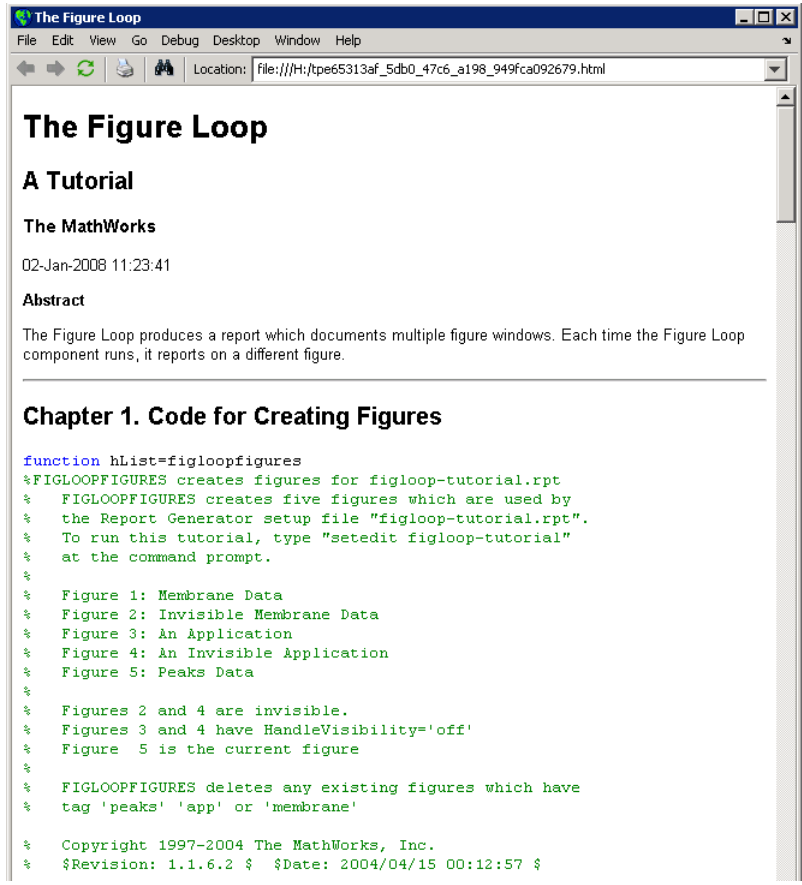
   The MATLAB Report Generator software generates MATLAB code for the `figloop_tutorial.rpt` report setup file. It saves this code in the `buildfigloop_tutorial.m` file in the folder you specify. Part of this file appears in the following figure.



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.


The Figure Loop
File Edit View Go Debug Desktop Window Help
Location: file:///H:/tpe65313af_5db0_47c6_a198_949fca092679.html
The Figure Loop
A Tutorial
The MathWorks
02-Jan-2008 11:23:41
Abstract
The Figure Loop produces a report which documents multiple figure windows. Each time the Figure Loop component runs, it reports on a different figure.
Chapter 1. Code for Creating Figures
function hList=figloopfigures
%FIGLOOPFIGURES creates figures for figloop-tutorial.rpt
% FIGLOOPFIGURES creates five figures which are used by
% the Report Generator setup file "figloop-tutorial.rpt".
% To run this tutorial, type "setedit figloop-tutorial"
% at the command prompt.
%
% Figure 1: Membrane Data
% Figure 2: Invisible Membrane Data
% Figure 3: An Application
% Figure 4: An Invisible Application
% Figure 5: Peaks Data
%
% Figures 2 and 4 are invisible.
% Figures 3 and 4 have HandleVisibility='off'
% Figure 5 is the current figure
%
% FIGLOOPFIGURES deletes any existing figures which have
% tag 'peaks' 'app' or 'membrane'
% Copyright 1997-2004 The MathWorks, Inc.
% $Revision: 1.1.6.2 $ $Date: 2004/04/15 00:12:57 $

Troubleshooting Report Generation Issues

In this section...
“Memory Usage” on page 4-25 “HTML Report Display on UNIX Systems” on page 4-26

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')
```

5

Add Content with Components

- "Components" on page 5-2
- "Report Structure Components" on page 5-4
- "Table Formatting Components" on page 5-5
- "Property Table Components" on page 5-6
- "Summary Table Components" on page 5-17
- "Logical and Looping Components" on page 5-21
- "Edit Figure Loop Components" on page 5-22

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 can interactively combine 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 3-11
- "Set Component Properties" on page 3-12

Use 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 5-4	Include a title page, chapters, sections, paragraphs, lists, tables, and other standard document structure elements.
"Table Formatting Components" on page 5-5	Organize generated content into tables.
"Property Table Components" on page 5-6	Display tables with property name/ property value pairs for objects.
"Summary Table Components" on page 5-17	Display tables with specified properties for objects.
"Logical and Looping Components" on page 5-21	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 stylesheet for HTML, Word, or PDF.

The output format and the associated template or stylesheet that you select for a report determines most aspects of the formatting of the generated report. For example, a report template or stylesheet typically uses different font sizes for chapter titles and section titles. For details, see "Report Output Format" on page 4-4.

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 following 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 strings of generated text.
List	Generate a list from a cell array of numbers or strings 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 strings.
Table	Parent a table body component. See "Table Formatting Components" on page 5-5.

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.

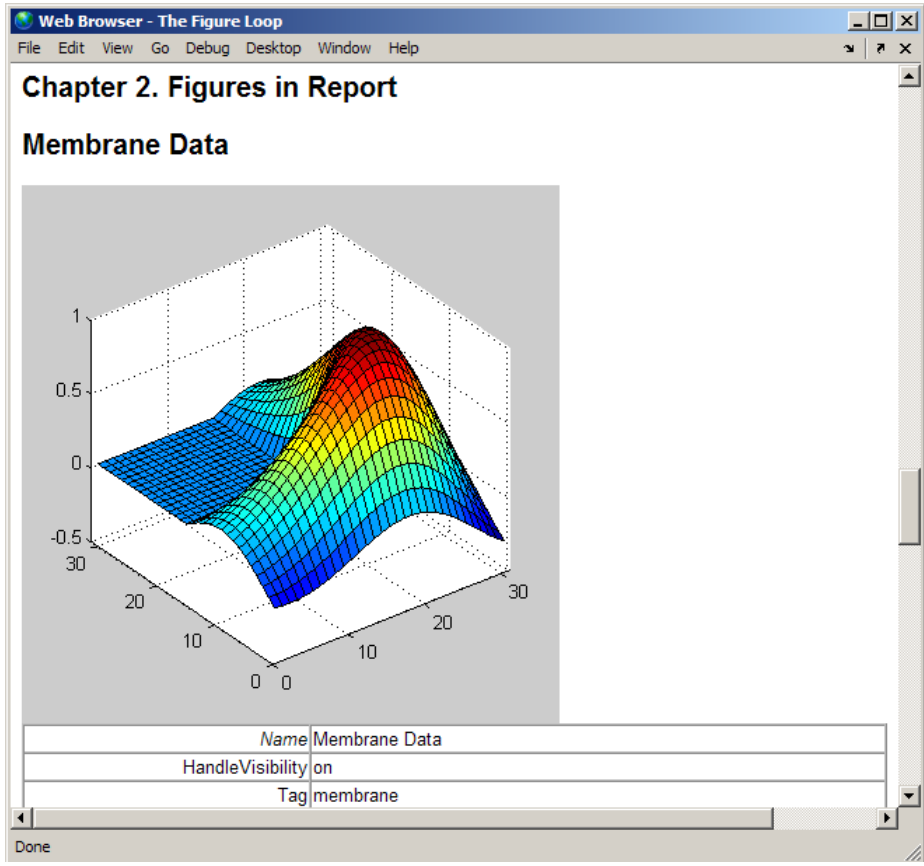
Tip Inserting a Table component into a setup also inserts all the descendant components required to generate a 2x2 table, creating a table template. Edit this template to create a table that suits your needs.

Property Table Components

In this section...
“About Property Table Components” on page 5-6
“Open the Example Report Template” on page 5-8
“Examine the Property Table Output” on page 5-8
“Select Object Types” on page 5-9
“Display Property Name/Property Value Pairs” on page 5-9
“Edit Table Titles” on page 5-12
“Enter Text into Table Cells” on page 5-12
“Add, Replace, and Delete Properties in Tables” on page 5-13
“Format Table Columns, Rows, and Cells” on page 5-14
“Zoom and Scroll” on page 5-16
“Select a Table” on page 5-16

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.

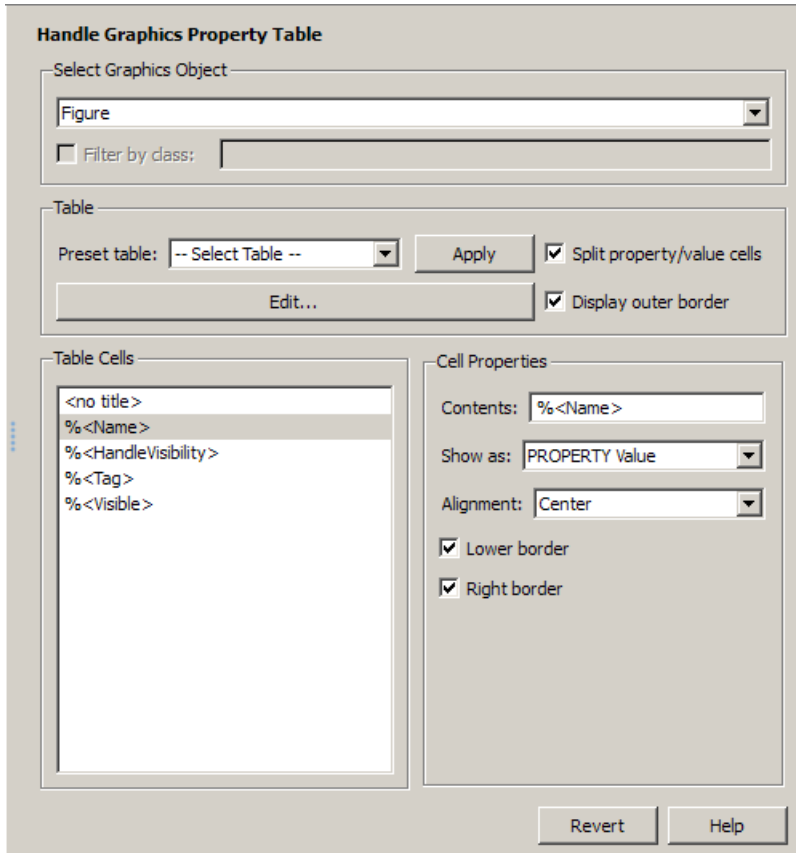
Open the Example Report Template

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

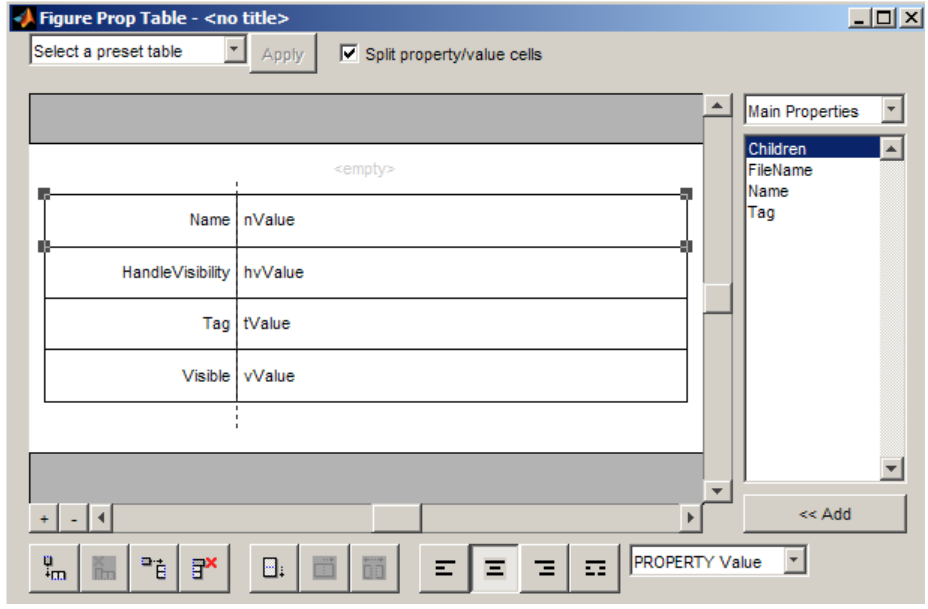
Property table components offer multiple object types on which to report. For example, the Handle Graphics Property Table lets you report on a figure, an axes object, or a Handle Graphics object.

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

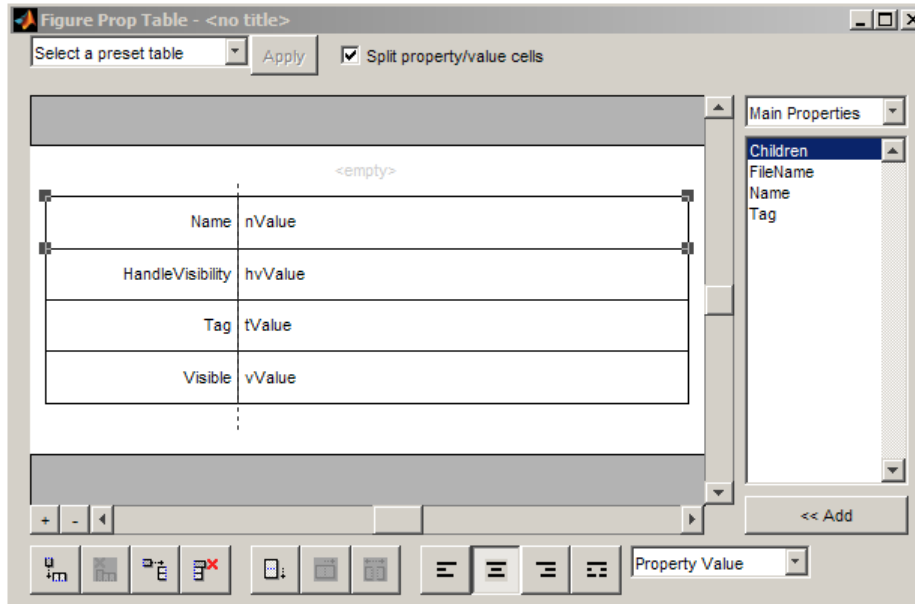
Split Property/Value Cells

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.



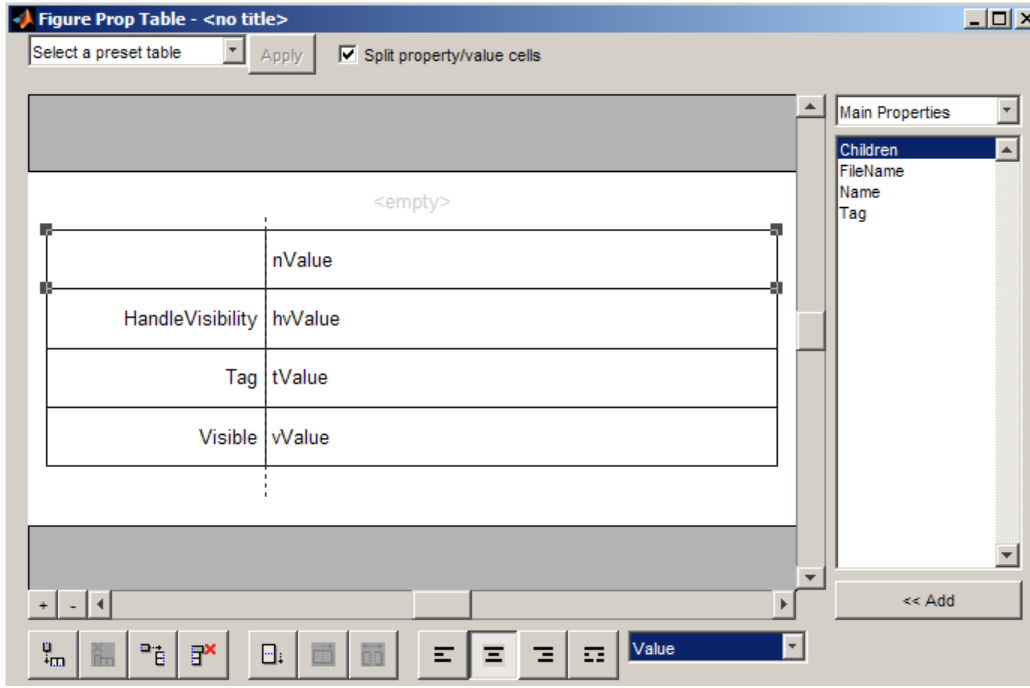
3. For the property name and property value to appear in adjacent horizontal cells in the table, select the **Split property/value cells** check box. The table is now in

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

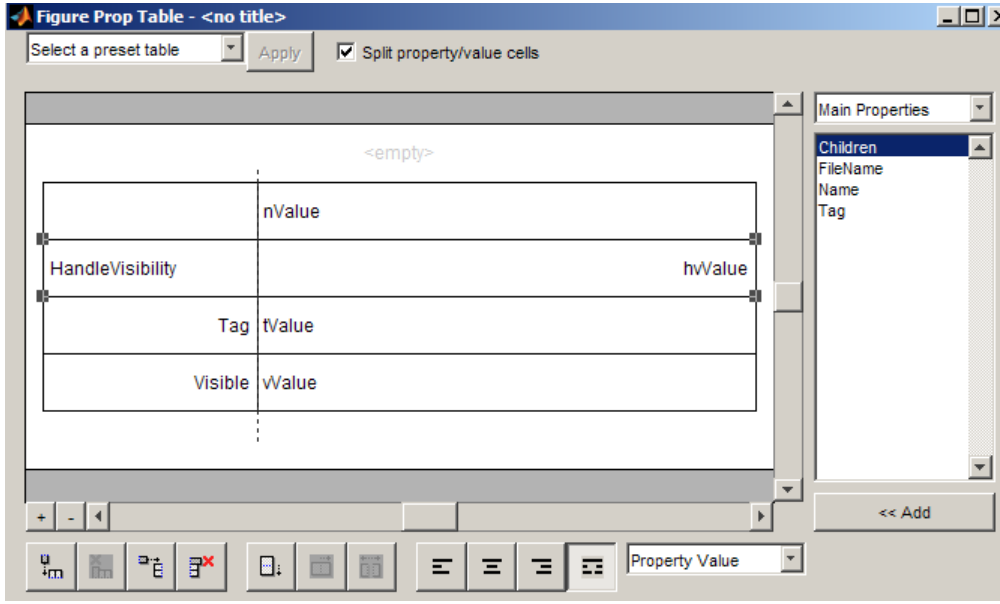
Property name/property value pairs can appear in cells in several ways. To specify how a given property name/property value pair appears in a cell, select that field in the table (for this tutorial, select the `Name` property). Choose `Value` from the display options drop-down list at the bottom of the 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

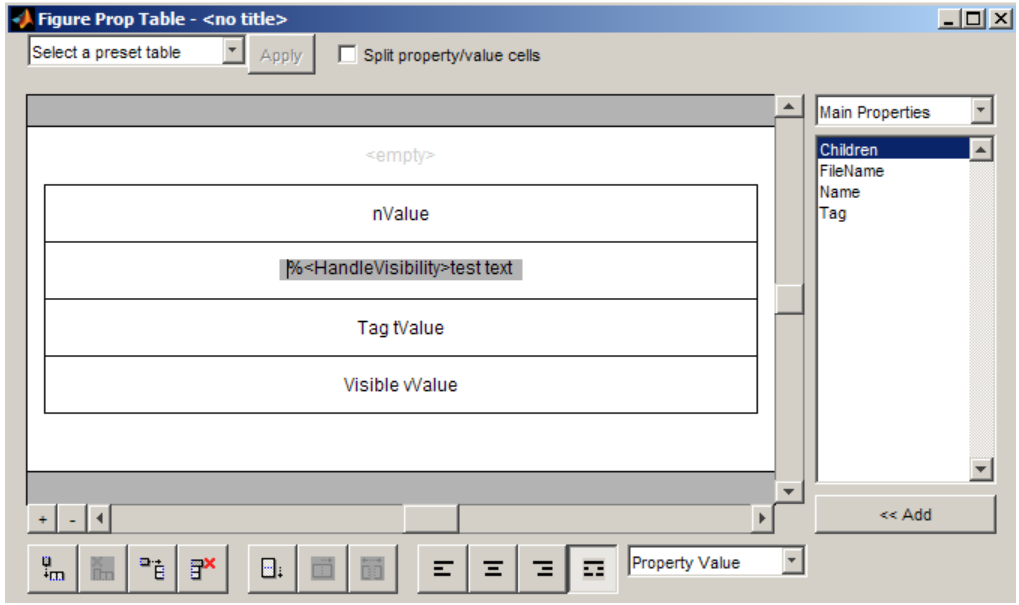
Table titles can contain properties and text. By default, the title of a table is the same as the value of the `%<Name>` property. You can modify this property to modify the table title.

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 table 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 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 5-12. For information about adding new table rows, see "Add and Delete Columns and Rows" on page 5-14.

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 5-13.

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)

Button	Action
	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
	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.

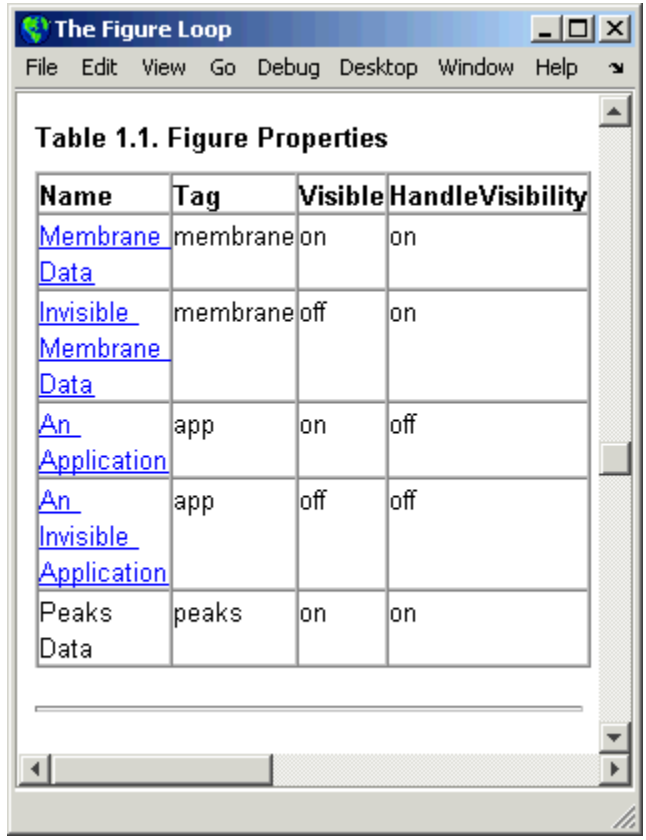
Note: You cannot save a custom table as a preset table. If you do so, you lose all changes to the custom table.

Summary Table Components

In this section...
“About Summary Table Components” on page 5-17
“Open the Example Report Template” on page 5-18
“Select Object Types” on page 5-19
“Add and Remove Properties” on page 5-19
“Set Relative Column Widths” on page 5-20
“Set Object Row Options” on page 5-20

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.



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:

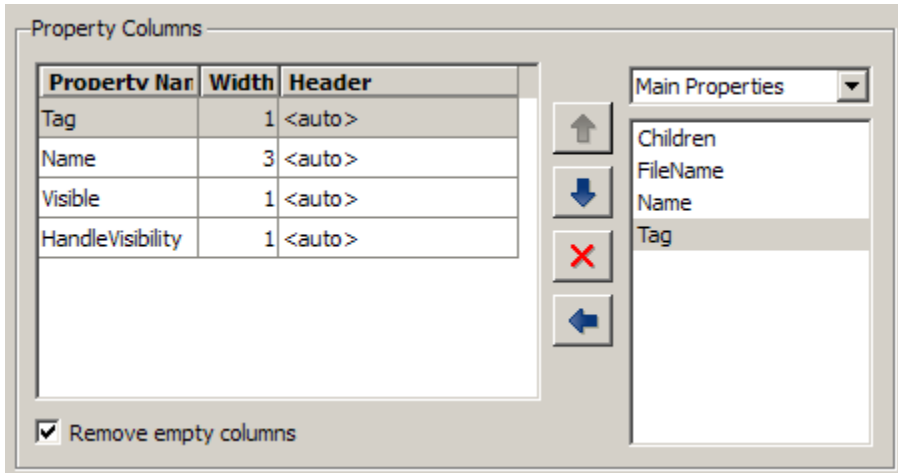
```
setedit figloop-tutorial
```

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. For more information, see "%<VariableName> Notation" on page 10-102 on the `Text` component reference documentation.

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 5-24
- "Loop on Visible Figures" on page 5-24
- "Loop on Figures with Tags" on page 5-24

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 5-22.

Edit Figure Loop Components

In this section...
“Figure Loop in a Report” on page 5-22
“Figure Properties” on page 5-23
“Loop on the Current Figure” on page 5-24
“Loop on Visible Figures” on page 5-24
“Loop on Figures with Tags” on page 5-24
“Modify Loop Section Options” on page 5-24

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.

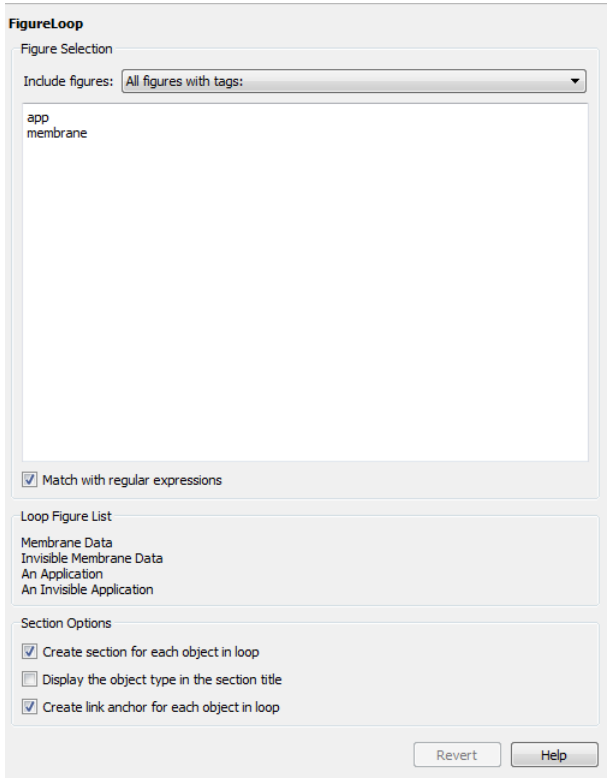


Figure Properties

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
Membrane Data	membrane	on	on
Invisible Membrane Data	membrane	off	on
An Application	app	on	off
An Invisible Application	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 5-13.

Loop on the Current Figure

To include only the current figure in the report, select `Current figure only` from the **Include figures** selection list. 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. In the **Include figures** selection list, select the `All figures with tags` option.
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.

6

Template-Based Report Formatting

- "Report Conversion Templates" on page 6-2
- "Generate a Report Using a Template" on page 6-4
- "Conversion Template Contents" on page 6-5
- "Copy a Conversion Template" on page 6-12
- "Open a Conversion Template" on page 6-13
- "Set Conversion Template Properties" on page 6-14
- "Move a Conversion Template" on page 6-15
- "Delete a Conversion Template" on page 6-16
- "Customize Microsoft Word Report Styles" on page 6-17
- "Customize Microsoft Word Part Templates" on page 6-20
- "Customize a Microsoft Word Title Page Template" on page 6-30
- "Create a Custom HTML or PDF Template" on page 6-36

Report Conversion Templates

In this section...
“Templates for Report Conversion” on page 6-2 “Customizing Templates” on page 6-2

Templates for Report Conversion

If you select a `from template` file format for output, the Report Generator uses an appropriate template to convert the XML content to the final format. For example, it uses a Microsoft Word template to format reports you generate in Word. The template determines the layout and format of the resulting document.

The **File format** option you select determines the template used to generate the report.

- `Direct PDF (from template)`. This format uses an HTML-based template that includes page layout elements that describe page size, headers, footers, and so on. Report generation using this option is faster than using `PDF (from Word template)`, especially for large documents that take a long time to open in Word.
- `PDF (from Word template)` . This format uses a Word template to convert the XML first to Word and then to PDF. Select this option if you want to use the formatting capabilities of Word as the basis of your PDF.
- `HTML (from template)`. This format uses an HTML template for the conversion.
- `Word (from template)`. This format uses a Microsoft Word template for the conversion.

Customizing Templates

The MATLAB Report Generator comes with default templates for each supported format. You can create customized versions of these templates to meet your report formatting and layout needs. After you create a custom template, you can select it from the template list next to the **File format** list when you set your report output options.

When you select a custom template to generate a report, you can apply styles in that template to instances of report components. For example, suppose your template defines three different styles of paragraph formatting. You can apply any of these styles to any Paragraph component instance in your report setup file. You can apply styles this way to any instance of a component that has a **Style Name** property.

Related Examples

- "Generate a Report Using a Template" on page 6-4
- "Copy a Conversion Template" on page 6-12
- "Customize Microsoft Word Report Styles" on page 6-17
- "Customize Microsoft Word Part Templates" on page 6-20
- "Customize a Microsoft Word Title Page Template" on page 6-30
- "Create a Custom HTML or PDF Template" on page 6-36

More About

- "Conversion Template Contents" on page 6-5

Generate a Report Using a Template

1. In Report Explorer, in the **Outline** pane, select the report.
2. In the Report Options dialog box that appears in the **Properties** pane, set the **File format** field to one of these options:
   - `Direct PDF (from template)`
   - `HTML (from template)`
   - `PDF (from Word template)`
   - `Word (from template)`
3. Optionally, from the list of templates available for the current file format, select a template.
4. If you select `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 .

More About

- "Report Conversion Templates" on page 6-2

Conversion Template Contents

In this section...
"Default Styles" on page 6-5
"Part Templates" on page 6-9
"Header and Footers in Word Conversion Templates" on page 6-10

A report conversion template contains:

- A main template with default style definitions for report elements.
- Part templates for the report elements such as title pages, chapters, and titles for sections and tables. Part templates contain fill-in-the-blanks holes for generated content.
- Headers and footers.

Default Styles

The default conversion 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, but not an HTML template.

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	Report Explorer Components the Style Formats
`MWSHComment`	MATLAB code comment
`MWSHKeywords`	MATLAB code keywords
`MWSHStrings`	MATLAB code strings
`rgAbstract`	`Title Page` component abstract
`rgAbstractTitle`	`Title Page` component abstract section
`rgAuthor`	`Title Page` component front page author
`rgAuthorVerso`	`Title Page` component back page author

Style	Report Explorer Components the Style Formats
rgBody	Text component
rgChapter	Chapter component
rgChapterTitle	Chapter component title
rgCopyright	Title Page component copyright
rgFigure	Paragraph that contains an image generated by a snapshot or Image component, to adjust the spacing of images relative to adjacent paragraphs
rgFigureCaption	The caption for Image component and snapshot components
rgFigureTitle	The Image component and snapshot components title
rgFigureTitleNumber	The Image component and snapshot components title number
rgFigureTitlePrefix	The Image component and snapshot components title prefix
rgFigureTitleText	The Image component and snapshot components title text
rgLegalNotice	Title Page component legal notice section
rgListStyle	Specifies the style of lists generated by the List component.
rgListTitle	The List component title
rgListTitleNumber	The List component title number
rgListTitlePrefix	The List component title prefix
rgListTitleText	The List component title text
rgParagraph	Paragraph component text
rgParagraphTitle	Paragraph component title

Style	Report Explorer Components the Style Formats
`rgProgramListing`	Code generated by: • `Text` component with **Show text as syntax highlighted MATLAB code** option is selected • `MATLAB Function Block` component • `Truth Table` component
`rgPubDate`	`Title page` report creation date
`rgPubDatePrefix`	`Title page` report creation date prefix
`rgSect1Title`	`Section` title for first-level section in a chapter
`rgSect1TitleNumber`	Number for `Section` title for first-level section in a chapter
`rgSect1TitlePrefix`	Prefix for `Section` title for first-level section in a chapter
`rgSect1TitleText`	Text for `Section` title for first-level section in a chapter
`rgSect2Title`	`Section` title for second-level section in a chapter
`rgSect2TitleNumber`	Number for `Section` title for second-level section in a chapter
`rgSect2TitlePrefix`	Prefix for `Section` title for second-level section in a chapter
`rgSect2TitleText`	Text for `Section` title for second-level section in a chapter
`rgSect3Title`	`Section` title for third-level section in a chapter
`rgSect3TitleNumber`	Number for `Section` title for third-level section in a chapter
`rgSect3TitlePrefix`	Prefix for `Section` title for third-level section in a chapter

Style	Report Explorer Components the Style Formats
rgSect3TitleText	Text for Section title for third-level section in a chapter
rgSect4Title	Section title for fourth-level section in a chapter
rgSect4TitleNumber	Number for Section title for fourth-level section in a chapter
rgSect4TitlePrefix	Prefix for Section title for fourth-level section in a chapter
rgSect4TitleText	Text for Section title for fourth-level section in a chapter
rgSect5Title	Section title for fifth-level section in a chapter
rgSect5TitleNumber	Number for Section title for fifth-level section in a chapter
rgSect5TitlePrefix	Prefix for Section title for fifth-level section in a chapter
rgSect5TitleText	Text for Section title for fifth-level section in a chapter
rgSubTitle	Title Page component subtitle
rgTable	Table content
rgTableTitle	Table title
rgTableTitleNumber	Table title number
rgTableTitlePrefix	Table title prefix
rgTableTitleText	Table title text
rgTitle	Title Page component front page title abstract, and legal notice section
rgTitleVerso	Title Page component back page title abstract, and legal notice section
rgTOCSection	Table of contents

Part Templates

The conversion templates include template parts to format specific elements of a report.

Part Template	Report Explorer Components the Part Template Formats
`rgRectoTitlePage`	`Title Page` Front title page contents, including the report title, subtitle, author, and an image.
`rgVersoTitlePage`	`Title Page` 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`	`Chapter/Section` Chapter (top-level section), including the title (prefix, such as `Chapter`, number, and title) and for the content.

`rgSect1Title`
 	
`rgSect2Title`
 	
`rgSect3Title`
 	
`rgSect4Title`
 	
`rgSect5Title`
	`Chapter/Section`
	
	Title for a section (sections below the chapter level). The title can include a prefix (such as `Chapter`), number, and title.

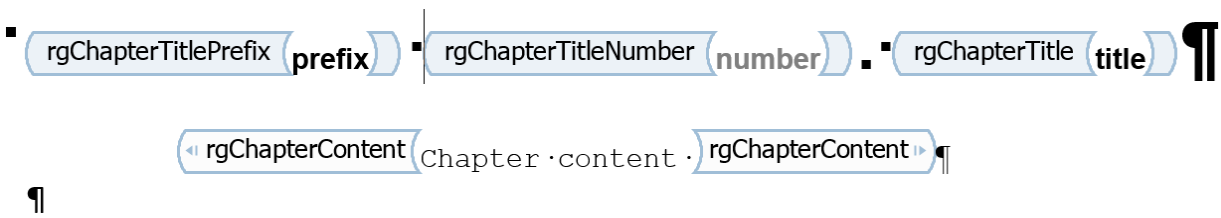
`rgListTitle`	`List` Title of the list.
`rgTableTitle`	`Table`, `Array-BasedTable`, and table components such as `Handle Graphics Property Table`

Part Template	Report Explorer Components the Part Template Formats
	Table title, including prefix (such as `Table`, number, and title) and for the content.
`rgFigureTitle`	`Table` and `Array-BasedTable` Table title, including prefix (such as `Table`, number, and title) and for the content.

Part Template Holes

Part templates include fill-in-the-blanks hole markup. The report converter fills holes with content that the MATLAB Report Generator generates.

For example, the `rgChapter` part template in the default Word conversion template includes an `rgChapterTitle` hole.



The report converter 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. If you add holes, the report converter ignores them.

Header and Footers in Word Conversion Templates

Word conversion templates include headers and footers for the document as a whole.

You can also specify headers and footers for the `rgRectoTitlePage`, `rgVersoTitlePage`, and `rgChapter` part templates.

Related Examples

- "Generate a Report Using a Template" on page 6-4
- "Copy a Conversion Template" on page 6-12
- "Customize Microsoft Word Report Styles" on page 6-17
- "Customize Microsoft Word Part Templates" on page 6-20
- "Customize a Microsoft Word Title Page Template" on page 6-30

More About

- "Report Conversion Templates" on page 6-2

Copy a Conversion Template

A template you create by copying a template appears in the list of templates. 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. The name in the list is not the file name you used when you saved the copy.

To change the template name that appears in the list of templates, in the Template Properties dialog box, specify the name in the **Display name** field. For details, see "Set Conversion Template Properties" on page 6-14.

Copy a Conversion Template

1. In Report Explorer, select **Tools** > **Edit Document Conversion Template**.
2. In the **Library** pane, select a template from the list. For example, select `Default Word Template`.
3. Click **Copy template**.
4. Navigate to where you want to save the template file. Select a folder that is on the MATLAB path (for example, in the `MATLAB` folder in your home folder).
5. Give the template a name and click **Save**.

Related Examples

- "Set Conversion Template Properties" on page 6-14
- "Open a Conversion Template" on page 6-13
- "Move a Conversion Template" on page 6-15
- "Delete a Conversion Template" on page 6-16
- "Customize Microsoft Word Report Styles" on page 6-17
- "Customize Microsoft Word Part Templates" on page 6-20
- "Customize a Microsoft Word Title Page Template" on page 6-30

More About

- "Report Conversion Templates" on page 6-2

Open a Conversion Template

You can open a copy of a conversion template to edit it. Opening a Word template opens the template in Microsoft Word. Opening an HTML or PDF template unzips the template and opens the main template document and the document part template library in the HTML editor specified by your Report Generator preferences (the MATLAB editor by default). To learn more about contents of templates, see "Conversion Template Contents" on page 6-5.

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 template from the list of templates in the **Library** pane. For example, select `Default Word Template`. The properties of the template appear in the Report Explorer **Properties** pane.
3. Click **Copy template** and follow the prompts to create a copy of the default template. If you already have a copy of a default template, you can skip this step.
4. Select the template copy and click **Open template**.

To also open the template style sheet, click **Open style sheet**.

Related Examples

- "Copy a Conversion Template" on page 6-12
- "Set Conversion Template Properties" on page 6-14

More About

- "Report Conversion Templates" on page 6-2

Set Conversion Template Properties

For copies of conversion templates, you can specify properties to describe the template. To learn more about using this dialog box, see “Copy a Conversion Template” on page 6-12 and “Open a Conversion Template” on page 6-13.

1. In Report Explorer, select **Tools > Edit Document Conversion Template**.
2. From the list of templates in the **Library** pane, click the template whose properties you want to set.
3. In the **Properties** pane, specify properties for the template.
   - **Template id** — Description of the template, such as `Word template for model reports`.
   - **Display name** — Template name to display in the list of templates (for example, `Model reports`).
   - **Description** — Description of the template.
   - **Creator** — Name of the person or organization that created the template.
4. Apply the properties by selecting another template in the list of templates.

Related Examples

- “Copy a Conversion Template” on page 6-12
- “Open a Conversion Template” on page 6-13
- “Move a Conversion Template” on page 6-15

More About

- “Report Conversion Templates” on page 6-2

Move a Conversion Template

You can change the location of a template file.

1. In Report Explorer, select **Tools > Edit Document Conversion Template**. The Template Browser appears in the Report Explorer.
2. In the Report Explorer, select the template to move.
3. In the Template Browser, select **Move template**.
4. In the file browser, navigate to the new location for the template file. Enter a file name, using the appropriate extension for the type of template (`.dotx` or `.htmtx`).
5. Click **Save**. The **Path** property in the Template Browser shows the new location.

Related Examples

- “Copy a Conversion Template” on page 6-12
- “Open a Conversion Template” on page 6-13
- “Delete a Conversion Template” on page 6-16

More About

- “Report Conversion Templates” on page 6-2

Delete a Conversion Template

1. In Report Explorer, select **Tools > Edit Document Conversion Template**. The Template Browser appears in the Report Explorer.
2. In the Report Explorer, select the template to delete.
3. In the Template Browser, select **Delete template** and click **Yes**.

Related Examples

- “Copy a Conversion Template” on page 6-12
- “Open a Conversion Template” on page 6-13
- “Move a Conversion Template” on page 6-15

More About

- “Report Conversion Templates” on page 6-2

Customize Microsoft Word Report Styles

In this section...
“Customize Default Microsoft Word Component Styles” on page 6-17 “Create Styles in a Microsoft Word Template” on page 6-17

You can customize report styles in a custom Word conversion 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: If you do not have a custom Word conversion template, see “Copy a Conversion Template” on page 6-12.

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 contains the style you want to customize.
3. In the **Properties** pane, click **Open stylesheet**. The Word Manage Styles dialog box appears.
4. Use the Manage Styles dialog box to modify or create styles.

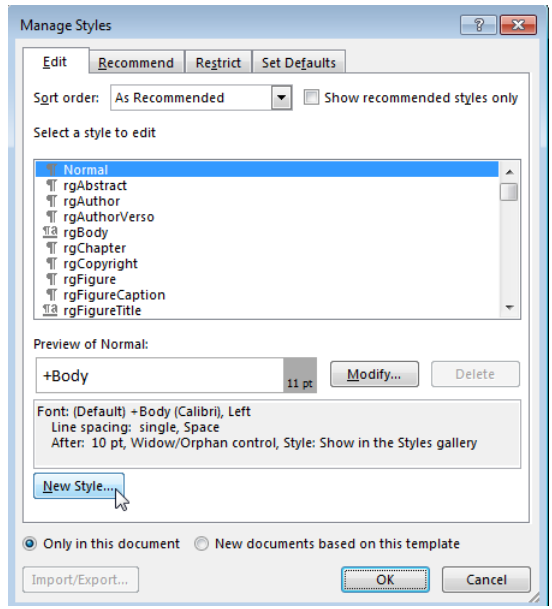
   Styles that begin with `rg` (for example, `rgParagraph`) are the default styles used for report components. A default style applies to all instances of a component with which it is associated. (In the Report Explorer, some components allow you to replace the name of a default style with the name a style that you create. This allows you to specify different styles for different instances of the same component.)

   For details, see “Create Styles in a Microsoft Word Template” on page 6-17.

Create Styles in a Microsoft Word Template

Note: If you do not have a custom Word conversion template, see “Copy a Conversion Template” on page 6-12.

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.
3. In the **Properties** pane, click **Open stylesheet**.
4. If applicable, select an existing style to use as a starting point for the new style.
5. Click the **New Style** button.



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.

Related Examples

- “Customize Microsoft Word Part Templates” on page 6-20
- “Customize a Microsoft Word Title Page Template” on page 6-30

More About

- “Report Conversion Templates” on page 6-2

- “Conversion Template Contents” on page 6-5

Customize Microsoft Word Part Templates

In this section...
"Custom Word Part Templates" on page 6-20
"Display the Developer Ribbon in Word" on page 6-21
"Customize a Word Conversion Part Template" on page 6-21
"Set Default Text Style for a Hole" on page 6-22
"Distinguish Inline and Block Holes" on page 6-24
"Avoid Changing Block Holes to Inline Holes" on page 6-25
"Delete a Hole" on page 6-25
"Add an Inline Hole" on page 6-27
"Add a Block Hole" on page 6-28
"Remove or Modify Chapter Prefix" on page 6-28

Custom Word Part Templates

You can create custom Word part templates (such as a title page part template) to:

- Tailor report formatting to meet your specific formatting requirements.
- Delete a hole. For a description of holes, see "Part Template Holes" on page 6-10.
- Rearrange the order of holes.
- Add fixed content to a footer or header.

If you delete a hole in a part template, then 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 only kind of holes that you can add to a part template are the holes that the Report Explorer supports for that part template. For example, for the `rgChapter` part template, you can only reinsert `rgChapterTitlePrefix`, `rgChapterTitleNumber`, `rgChapterTitle`, and `rgChapterContent` holes. Do not add multiple instances of the same kind of hole in a part template.

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 Conversion Part Template

To customize a report element such as a title page, replace the appropriate default part template with a customized copy of the part template. For another example illustrating how to create a custom Word part template, see "Customize a Microsoft Word Title Page Template" on page 6-30.

Note: If you do not have a custom Word conversion template, see "Copy a Conversion Template" on page 6-12.

---

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.
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 `mlreportgen`. 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.

Note: If you do not have a custom Word conversion template, see "Copy a Conversion Template" on page 6-12.

---

1 In the Report Explorer, select **Tools > Edit Document Conversion Template**.

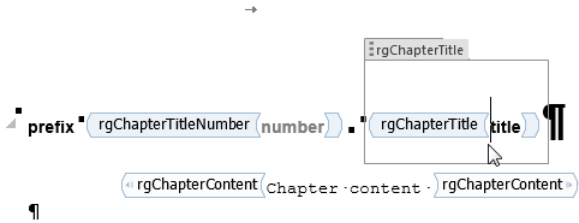
2 In the list of templates in the middle pane, select the custom template that has the hole you want to set the default text style for.

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.



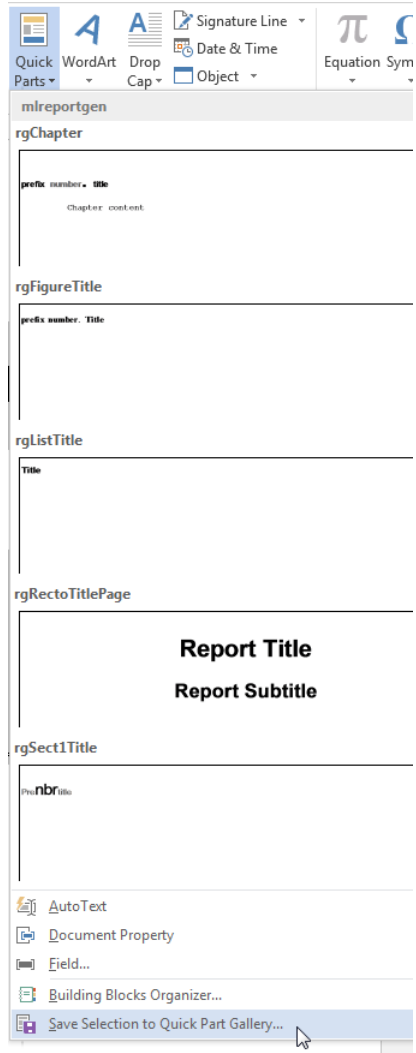
7 Select **Properties**.

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 `mlreportgen`. 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.

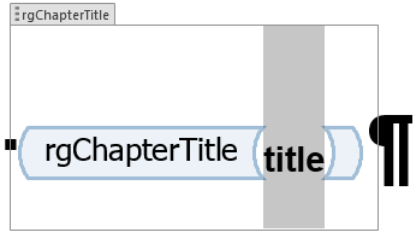
You can configure the Word editor to provide visual cues that indicate whether a hole is an inline or block hole.

Note: If you do not have a custom Word conversion template, see "Copy a Conversion Template" on page 6-12.

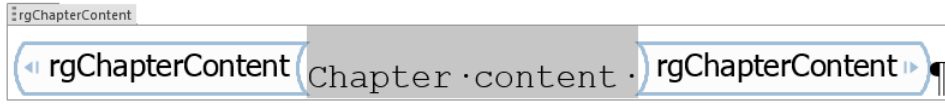
---

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 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.



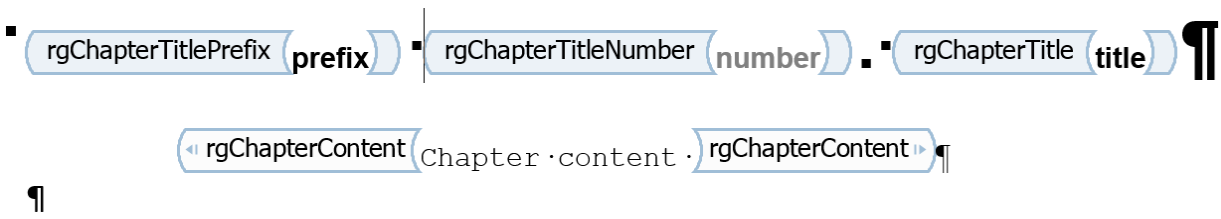
    - Block hole — The bounding box does includes the paragraph marker.



Avoid Changing Block Holes to Inline Holes

Do not change a block hole to an inline hole.

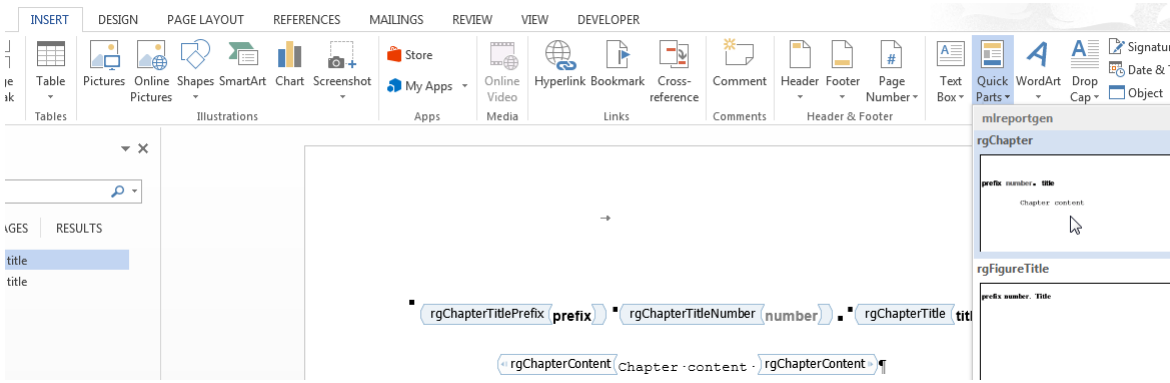
You can accidentally change a block hole to an inline hole by removing the paragraph marker of an inline hole that is followed by a block hole. For example, if you delete the paragraph marker for the `rgChapterTitle` inline hole, the `rgChapterContent` block hole changes to an inline hole.



Delete a Hole

Note: If you do not have a custom Word conversion template, see "Copy a Conversion Template" on page 6-12.

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 `mlreportgen`. 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

The only kind of holes that you can add to a part template are the holes that the Report Explorer supports for that part template. For example, for the `rgChapter` part template, the only inline holes that you can reinsert are `rgChapterTitlePrefix`, `rgChapterTitleNumber`, and `rgChapterTitle` holes. Do not add multiple instances of the same kind of hole in a part template.

Note: If you do not have a custom Word conversion template, see "Copy a Conversion Template" on page 6-12.

---

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 Aa. 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 `mlreportgen`. 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

The only kind of holes that you can add to a part template are the holes that the Report Explorer supports for that part template. For example, for the `rgChapter` part template, the block hole that you can reinsert is `rgChapterContent` holes. Do not add multiple instances of the same kind of hole in a part template.

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 If you have not done so, create a custom Word template. See “Create a Custom Template” on page 6-30.

2 Open the template.

3 On the Word **Insert** ribbon, in the **Text** area, click the **Explore Quick Parts** button.

4 To insert an instance of the part you want to modify in your template, select the **rgChapter** quick part.

5 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**.

6 Select the edited instance. Then, on the **Insert** ribbon, click the **Explore Quick Parts** button and select **Save Selection to Quick Parts Gallery**.

7 In the dialog box, set **Name** to `rgChapter` and **Category** to `mlreportgen`, and then click **OK**. Confirm that you want to overwrite the previous version.

8 Save and close the template.

Related Examples

- "Copy a Conversion Template" on page 6-12
- "Customize Microsoft Word Report Styles" on page 6-17
- "Customize a Microsoft Word Title Page Template" on page 6-30

More About

- "Report Conversion Templates" on page 6-2
- "Conversion Template Contents" on page 6-5

Customize a Microsoft Word Title Page Template

In this section...
“Create a Custom Template” on page 6-30
“Change the Color of a Report Title” on page 6-31
“Assign the Template to a Report” on page 6-33
“Customize Title Page Content and Layout” on page 6-34

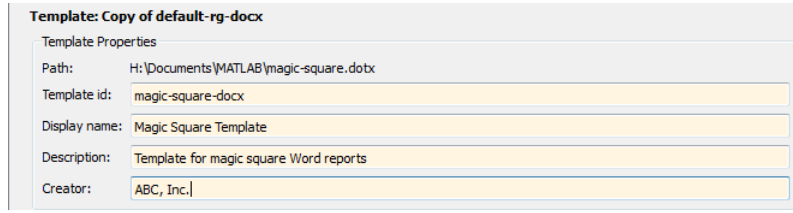
The Report Explorer default Word document conversion template contains document part templates for the front (recto) and back (verso) side of a report title page. The Report Explorer file converter for the `Word (from template)` output type uses the title page part templates to produce the title pages 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 6-31.

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

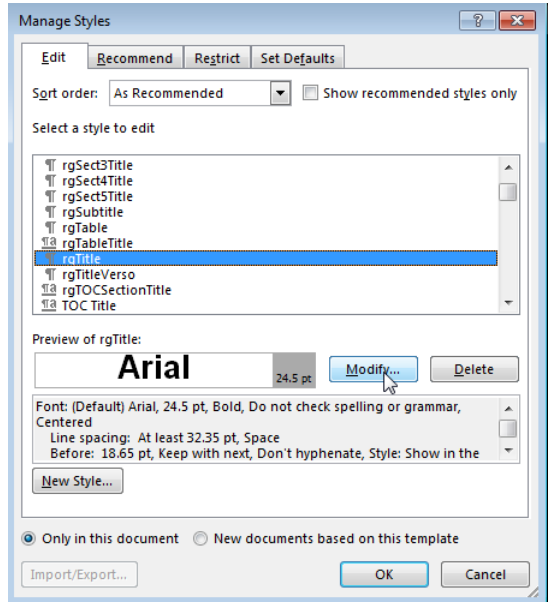
You can customize the `Magic Square Template` (see "Create a Custom Template" on page 6-30) to use blue text for the report title.

1 In the Report Explorer, select **Tools** > **Edit Document Conversion Template**.

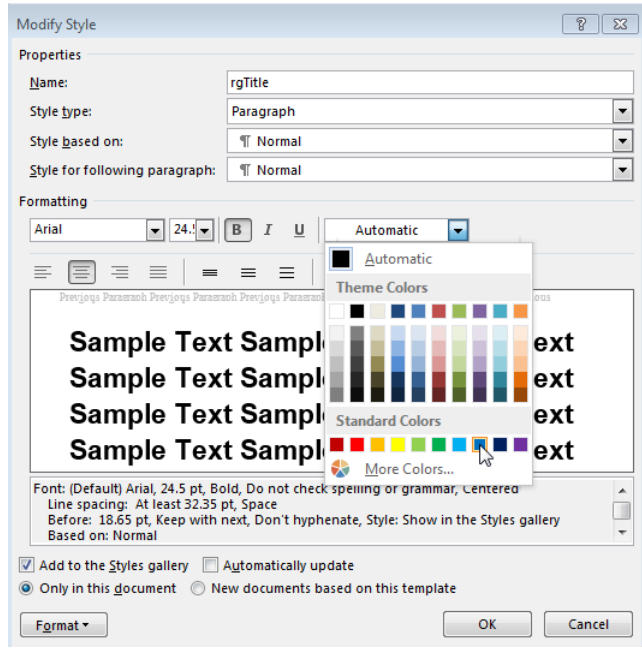
2 In the Report Explorer list of Word templates, select `Magic Square Template`.

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**.

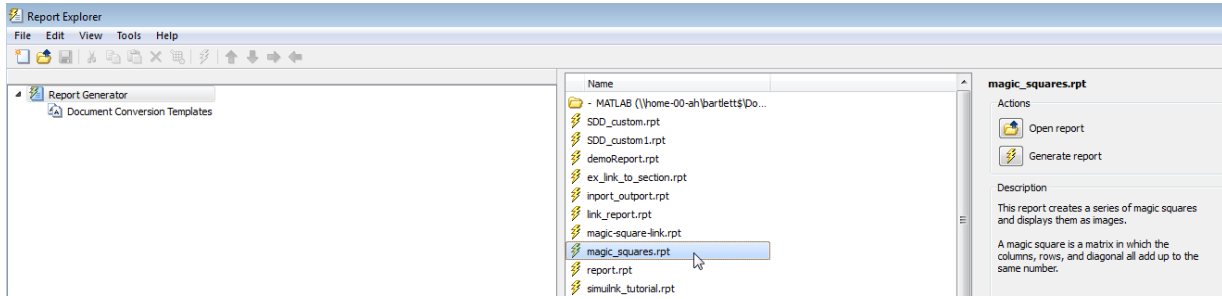
7 Save and close the template.

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`.

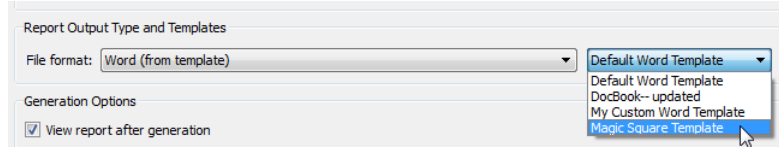


3 In the Report Options pane, click **Open report**.

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 6-30). 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.

Related Examples

- “Copy a Conversion Template” on page 6-12
- “Customize Microsoft Word Report Styles” on page 6-17

More About

- “Report Conversion Templates” on page 6-2
- “Conversion Template Contents” on page 6-5

Create a Custom HTML or PDF Template

In this section...
“Copy the Template” on page 6-36
“Assign the Template to a Report” on page 6-36
“Select an HTML Editor” on page 6-37
“Edit HTML or PDF Templates” on page 6-38
“Edit HTML or PDF Styles in a Template” on page 6-38

Copy the Template

To customize the format styles used in the default HTML or PDF template, copy the template and modify or add style definitions in the copy.

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).

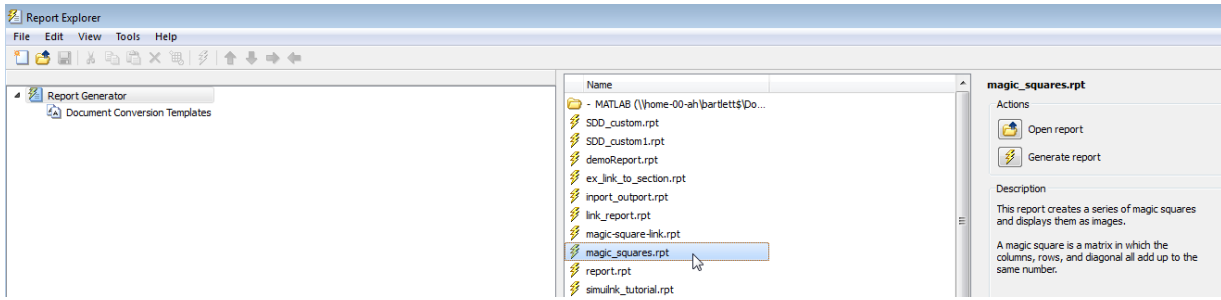
   Specify the file name, using the default file extension for an HTML template (`.htmtx`) or PDF template (`.pdftx`). Click **Save**.
5. In the list of templates in the middle pane, select the template copy.
6. In the Properties pane, in the **Template id** and **Display name** fields, specify a unique 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.
7. To save the template properties you entered, click outside of the Properties pane.

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 stylesheet, the Report Explorer replaces `FileName` with the template that you selected. The ampersand (&) opens the editor in the background.

Edit HTML or PDF Templates

The templates consist of the main part (`root.html`), which defines the default page, and the document part templates (`docpart_templates.html`). You can make similar types of changes in these templates as you can in Word templates. See "Customize Microsoft Word Part Templates" on page 6-20.

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 13-43.
- PDF templates can use only a subset of standard HTML elements. See "Standard HTML Elements" on page 13-45.

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.

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 Stylesheets" on page 4-7. 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 Properties 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.

Related Examples

- "Generate a Report Using a Template" on page 6-4
- "Customize Microsoft Word Report Styles" on page 6-17
- "Customize Microsoft Word Part Templates" on page 6-20

More About

- "Report Conversion Templates" on page 6-2

External Websites

- FO Summary

7

Create Custom Components

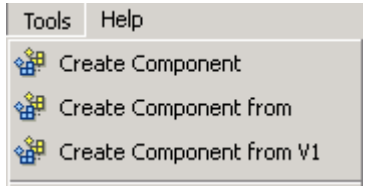
- "Create Custom Components" on page 7-2
- "Define Components" on page 7-5
- "Specify Tasks for a Component to Perform" on page 7-12
- "Customized Components" on page 7-18

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:

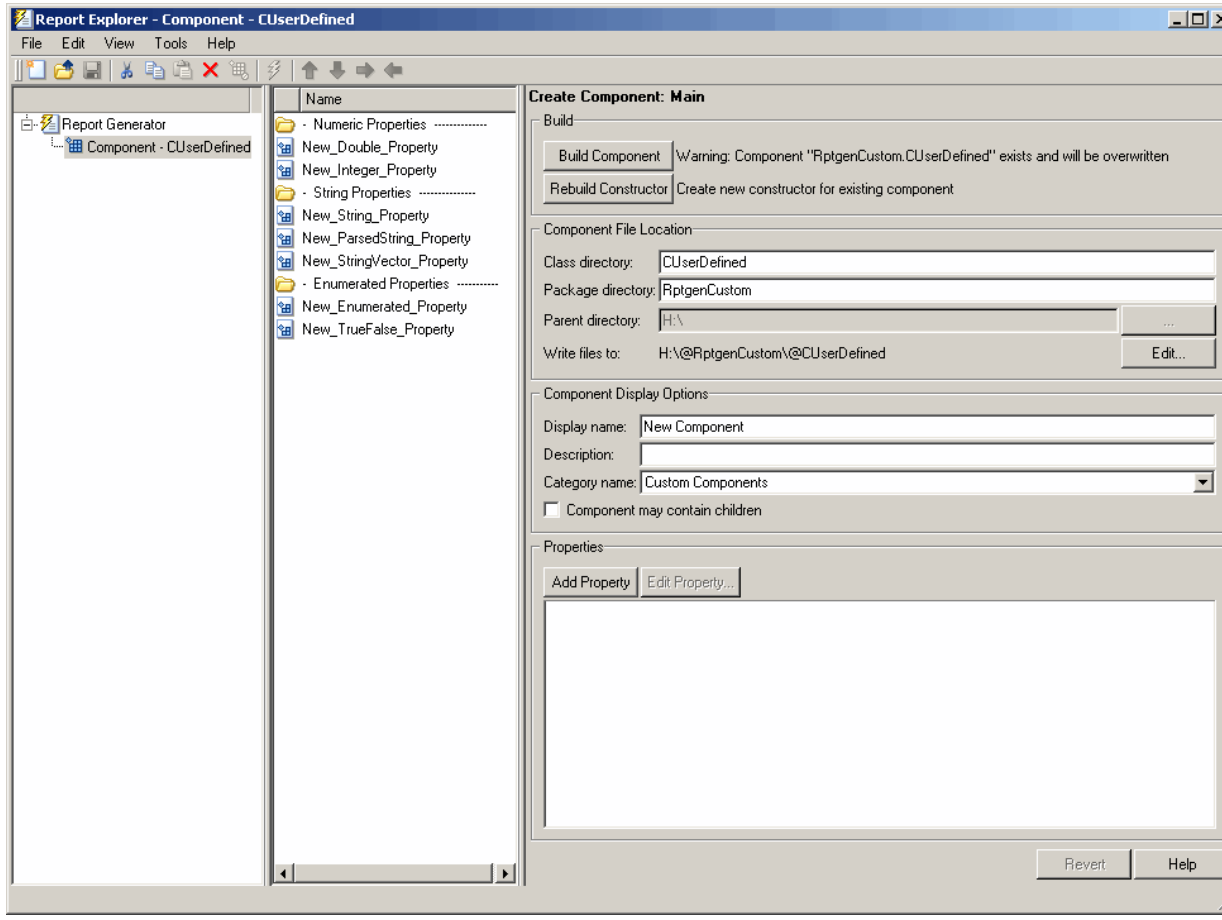


- To create a custom component, select **Create Component**.
- To create a custom component from an existing component, select **Create Component from**.
- To create a component from an existing version 1 component, select **Create Component from V1**.

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.

Related Examples

- “Build Components” on page 7-10
- “Specify Tasks for a Component to Perform” on page 7-12
- “Define Components” on page 7-5

Define Components

In this section...
“Required Component Data” on page 7-5
“Specify the Location of Component Files” on page 7-5
“Set Component Display Options” on page 7-6
“Specify Component Properties” on page 7-8
“Modify Existing Components” on page 7-10
“Build Components” on page 7-10
“Rebuild Existing Components” on page 7-11
“Remove a Component” on page 7-11

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 7-5.
2. Properties of the component. For more information, see “Specify Component Properties” on page 7-8.
3. Display options for the component, including its display name, category, and description. For more information, see “Set Component Display Options” on page 7-6.

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.

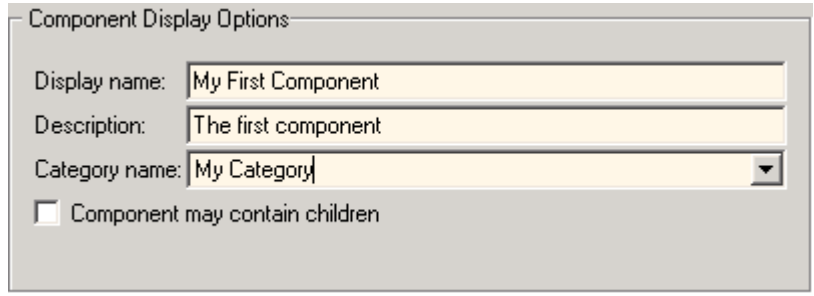
Set Component Display Options

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**.

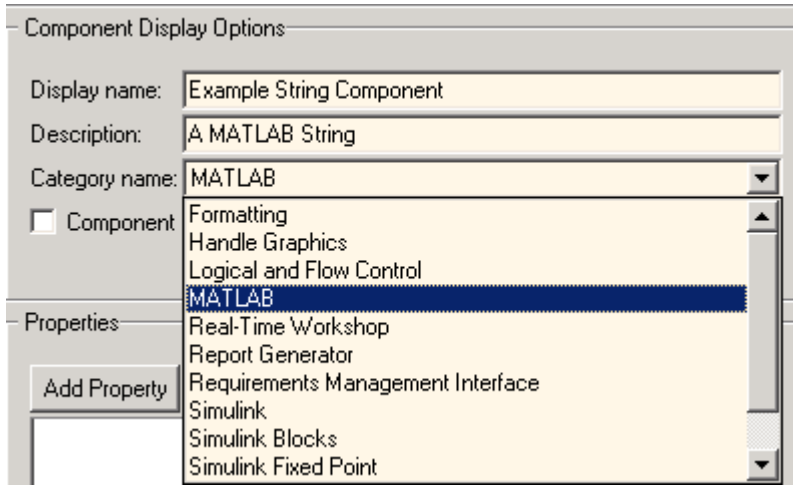
   The following example shows how to create a component called `My First Component` in a category called `My Category`.



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.



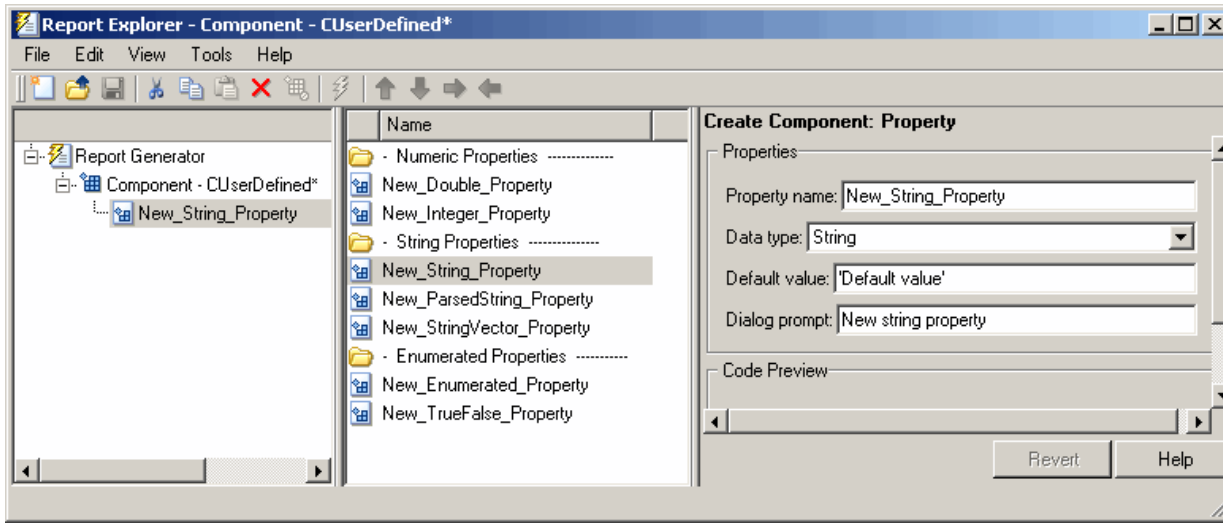
To create a custom component category, type the name for the category into the **Category name** field. This category name appears in the list of available categories in the Report Explorer.

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 following figure displays the property values for `New_String_Property`.

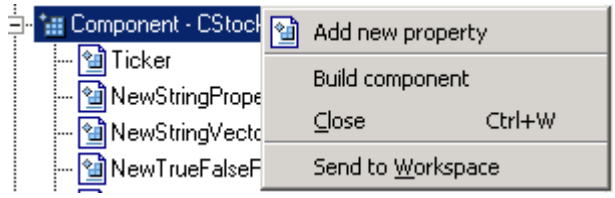


Add Properties to Components

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:

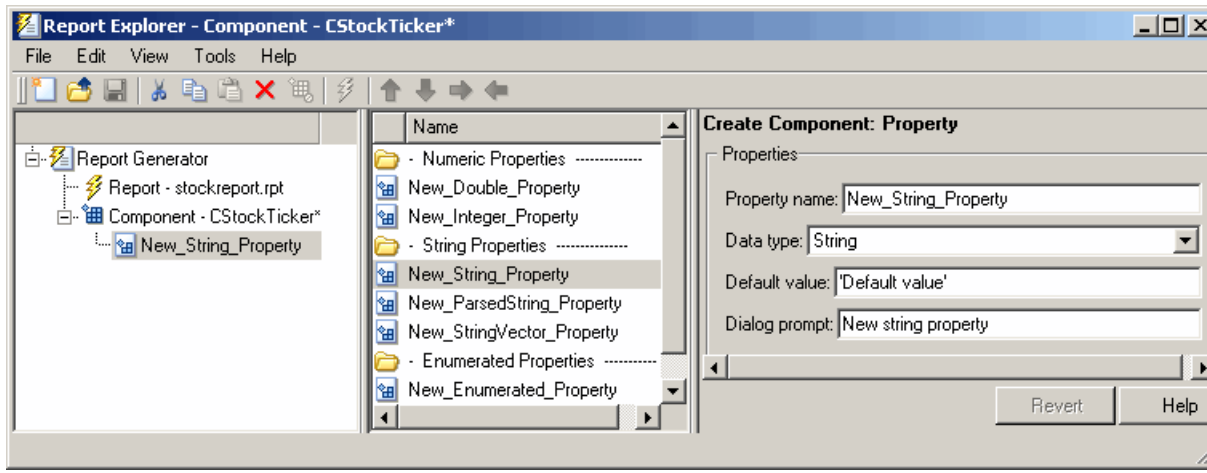
1. Right-click the name of the component to which you want to add properties in the Outline pane on the left. Select **Add new property** from its context menu.



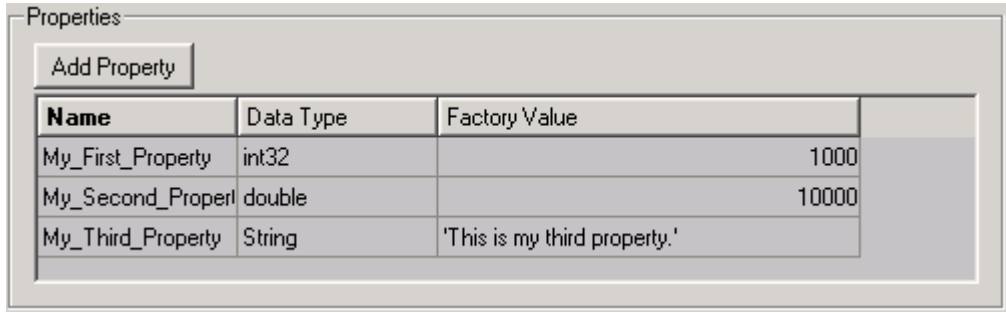
2 Right-click the name of a predefined property in the Options pane in the middle. From the context menu, select **Add property**.

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`
   - `String`
   - `String Vector`
   - `%<Parsed String>`

     Use this data type to include the value of a variable in the MATLAB workspace in a component. For more information about this data type, see "%<VariableName> Notation" on page 10-102 on the `Text` reference page.
   - `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 7-5.

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 7-12

"Required Customization: Specify Format and Content of Report Output" on page 7-12

"Change a Component's Outline String in the Report Explorer Hierarchy" on page 7-14

"Modify the Appearance of Properties Dialog Boxes" on page 7-15

"Specify Additional Component Properties" on page 7-16

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 7-12
- "Change a Component's Outline String in the Report Explorer Hierarchy" on page 7-14
- "Modify the Appearance of Properties Dialog Boxes" on page 7-15
- "Specify Additional Component Properties" on page 7-16

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 text string:

```
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 String 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 string 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 string. If the data is empty, the second argument is the return value, The third argument is the maximum allowed size of the resulting string.

```
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)
```

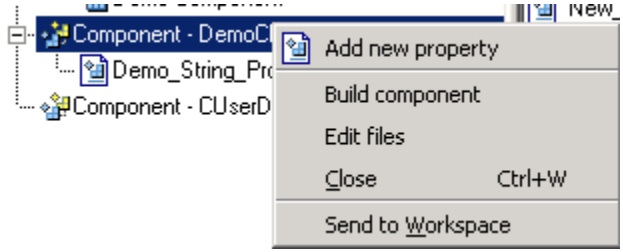
Where:

- `thisComp` is the instance of the component being edited.
- `name` is a string 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 7-16
- "Modify a Component Description" on page 7-16
- "Change a Component Display Name" on page 7-17
- "Change a Component Category Name" on page 7-17
- "Register Components" on page 7-17
- "Display Component Help in the MATLAB Help Browser" on page 7-17

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 n `getDescription.m` is the same value as the **Description** field in the Report Explorer. The following example shows how to edit the `compDesc` string 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` string 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` string 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**.

Customized Components

In this section...
“Fetching Securities Data and Displaying It in a Table” on page 7-18
“Displaying Securities Data in Two Tables” on page 7-23

Note: These examples require the Datafeed Toolbox™ software.

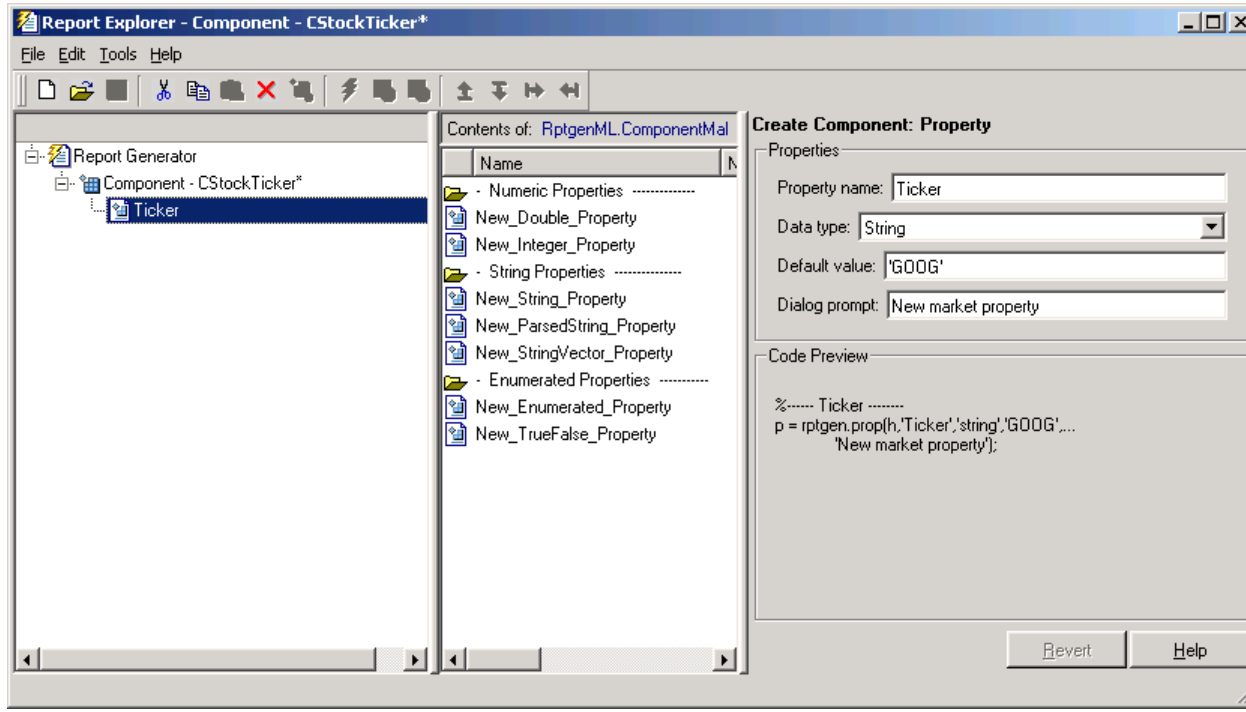
Fetching Securities Data and Displaying It in a Table

This example shows how to create a component that fetches securities data and displays it in a report as a table.

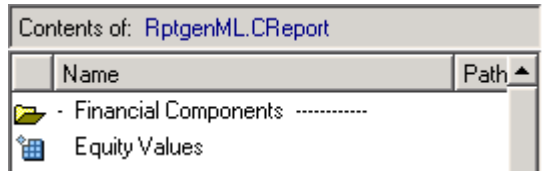
1. In the class folder named `CStockTicker`, insert a component named `Equity Values`.
2. Give the component one string property named `Ticker`, and specify its attributes.
   a. In the Options pane of the Report Explorer, double-click **New_String_Property**.
   b. For **Property name**, specify a valid MATLAB variable name.
   c. Specify the property's data type. In this case, `Ticker` is a string value, which is the default data type.
   d. Specify the property's default value.

      Because this example fetches ticker values for the security `Google`, set the **Default value** to `'GOOG'`. (The single quotation marks are required to specify a string value for this field.)

      Your specified settings appear in the **Code Preview** pane.



3 To build the new component, click the **Build** button in the Report Explorer. The Equity Values component now appears in the Options pane in the middle of the Report Explorer.



4 Edit the component's `execute.m` file to retrieve stock market data and display it in a table in the generated report.

a In the `@CStockTicker` folder, open `execute.m`.

b Enter the following text into `execute.m`.

```
function out=execute(thisComp,parentDoc,varargin)
```

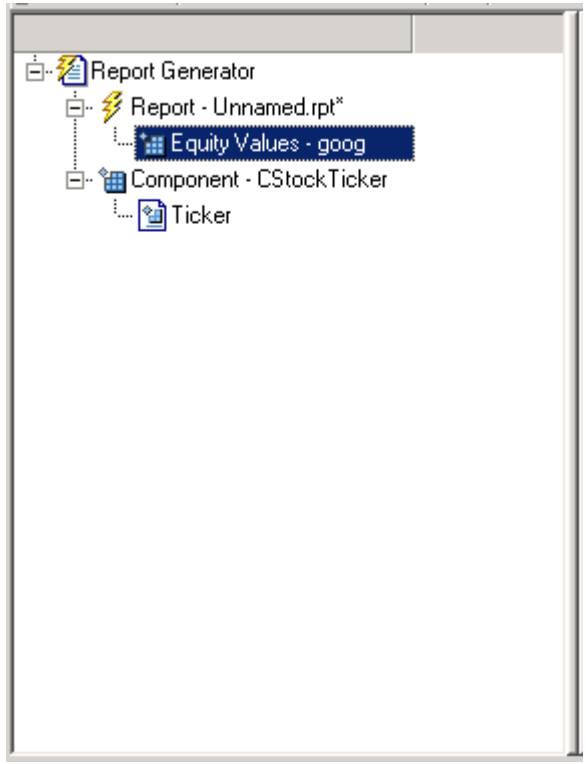
```
stockQuote = fetch(GOOG, thisComp.Ticker);
stockQuote.Date = datestr(stockQuote.Date,1);
stockQuote.Time = datestr(stockQuote.Time,13);
out = execute(rptgen.cfr_table(...
     'Source', stockQuote,...
     'numHeaderRows', 0,...
     'TableTitle', 'Stock Market Pricing Data'),...
    parentDoc);
```

5 Append the security symbol, `goog`, to the component name. Enter the following text into `getOutlineString.m`.

```
function olstring=getOutlineString(thisComp)

olstring = [getName(thisComp),' - ',thisComp.Ticker];
```

The component name now appears as `Equity Values – goog`.



6 Modify the `getdialogschema.m` file to change the appearance of the Properties pane. Enter the following text into this file to display the last quoted price for the security in the Properties pane.

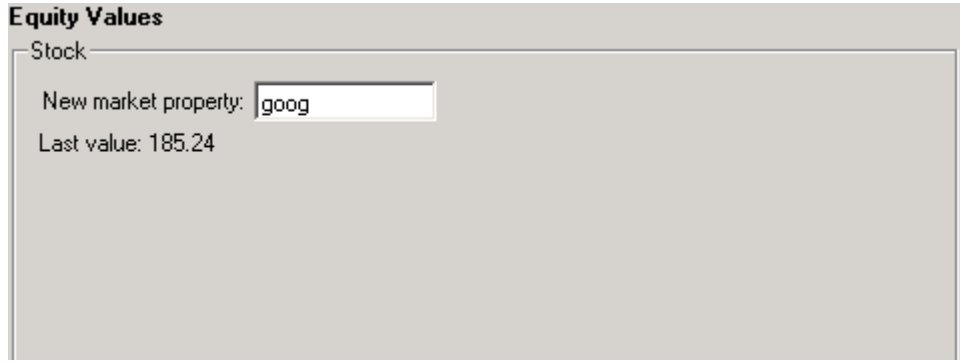
```
function dlgStruct = getdialogschema(thisComp, name)

try
    currQuote = fetch(yahoo, thisComp.Ticker);
    quoteStr = sprintf('Last value: %g', currQuote.Last);
catch
    quoteStr = sprintf('Warning: ...
    "%s" is not a valid symbol.', thisComp.Ticker);
end

dlgStruct = thisComp.dlgMain(name,...
    thisComp.dlgContainer({
```

```
    thisComp.dlgWidget('Ticker',...
      'DialogRefresh',true,...
      'RowSpan',[1 1],'ColSpan',[1 1]);
    thisComp.dlgText(quoteStr,...
      'RowSpan',[2 2],'ColSpan',[1 1]);
  },'Stock',...
    'LayoutGrid',[3 2],...
    'RowStretch',[0 0 1],...
    'ColStretch',[0 1]));
```

The Properties pane for the component, `Equity Values`, now looks as follows.



7 Click **File** > **Report** to generate the report. The following output appears in the report.

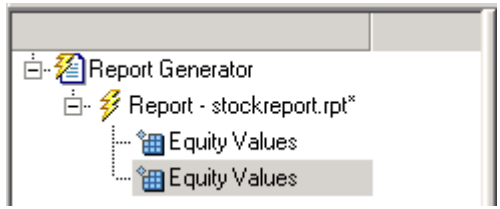
Table 1. Stock Market Pricing Data

Symbol	GOOG
Last	185.25
Date	15-Nov-2004
Time	15:20:00
Change	3.25
Open	180.45
High	188.32
Low	178.75
Volume	10651060

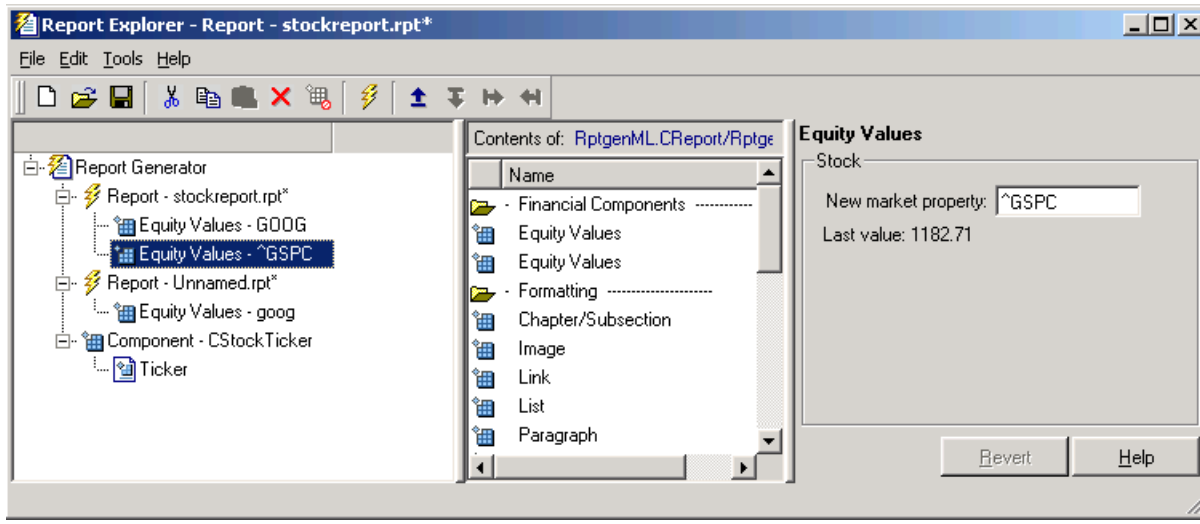
Displaying Securities Data in Two Tables

This example, which shows how to use multiple properties within a component, expands upon "Fetching Securities Data and Displaying It in a Table" on page 7-18.

1 Create a report setup file and save it as `stockreport.rpt`. Add two `Equity Values` components to the setup file.



2 Edit the entry in the **New marker property** field to change the `ticker` property of the second component to `'^GSPC'` (S&P 500 index).



3 Run the report.

The report displays two tables of data, one for Google® and another for the S&P 500 index.

Table 1. Stock Market Pricing Data

Symbol	GOOG
Last	185.25
Date	15-Nov-2004
Time	15:20:00
Change	3.25
Open	180.45
High	188.32
Low	178.75
Volume	10651060

Table 2. Stock Market Pricing Data

Symbol	^GSPC
Last	1183.13
Date	15-Nov-2004
Time	15:35:00
Change	-1.04
Open	1183.81
High	1184.48
Low	1179.85
Volume	1503245440

8

Create Custom Stylesheets

- "Stylesheets" on page 8-2
- "Create a New Stylesheet" on page 8-4
- "Edit, Save, or Delete a Stylesheet" on page 8-5
- "Edit Stylesheet Data Items" on page 8-9
- "Stylesheet Cells for Headers and Footers" on page 8-23
- "Customized Stylesheets" on page 8-29
- "Configure PDF Fonts" on page 8-40

Stylesheets

In this section...
"Built-In Versus Custom Stylesheets" on page 8-2
"Customize Stylesheets Using Data Items" on page 8-3

Built-In Versus Custom Stylesheets

Stylesheets specify formatting and display settings for reports. The report-generation process uses stylesheets 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 stylesheet.

The following table lists report output formats and their default stylesheets.

Report Format	Default Stylesheet
HTML	Uses stylesheets for either single- or multiple-page documents
PDF	Formatting Object (FO) stylesheet
RTF, Word	Document Style Semantics and Specification Language (DSSSL) stylesheet

The following table shows a list of properties for the built-in stylesheets.

Properties of Stylesheets

Name	Description
Description	A description of the stylesheet.
Display name	The stylesheet name that appears in the Options pane.
Transform type	The process used to generate reports that use a specified stylesheet. Supported types are: • HTML • FO (Formatting Object) for PDF reports • DSSSL (Document Style Semantics and Specification Language) for RTF and Word reports

Name	Description
	Note: This field is not editable.

In most cases, the stylesheets provided with the MATLAB Report Generator software should be more than adequate for your needs. However, you may want to modify the built-in stylesheets to meet special requirements. For example, suppose one of the built-in stylesheets meets your requirements, but you want to change the page orientation. You can create a custom stylesheet by editing the built-in stylesheet to your specifications.

Customize Stylesheets Using Data Items

Each built-in stylesheet 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 stylesheets 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 Stylesheet Data Items” on page 8-9.

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 Stylesheet

To create a stylesheet:

1. Open the Report Explorer.
2. From the menu bar, click **Tools > Edit Stylesheet**.
3. In the Properties pane on the right, choose the built-in stylesheet for the format with which you want to work. Options are:

   - **New HTML.** Creates a stylesheet for HTML reports.
   - **New multi-page HTML.** Creates a stylesheet for HTML reports with more than one page.
   - **New FO (PDF).** Creates a stylesheet for PDF reports.
   - **New DSSSL (RTF).** Creates a stylesheet for RTF reports.

   The new stylesheet appears in the Outline pane on the left.
4. In the Properties pane on the right, modify the properties for the stylesheet as needed. Add data items to the new stylesheet:

   a. Drag the data item you want to add from the Options pane in the middle to the stylesheet 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 Stylesheet Data Items” on page 8-9
5. Save the stylesheet. For information about how to save a stylesheet, see “Save a Stylesheet” on page 8-7.

Edit, Save, or Delete a Stylesheet

In this section...
“Edit a Stylesheet” on page 8-5
“Save a Stylesheet” on page 8-7
“Delete a Stylesheet” on page 8-8

Edit a Stylesheet

To edit a stylesheet:

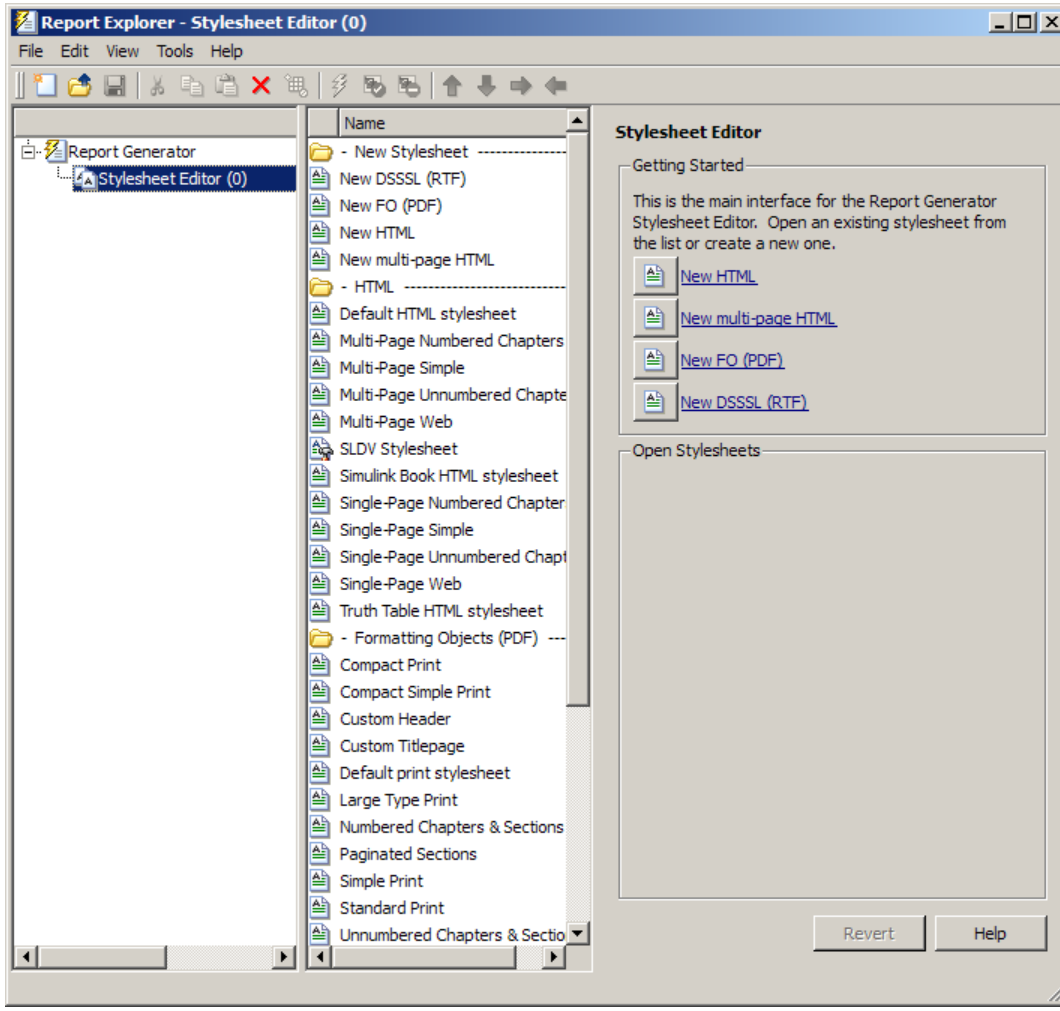
1. In Report Explorer, select a report setup file in the Outline pane on the left.
2. From the menu bar, click **Tools > Edit Stylesheet**.

The Report Explorer displays as follows.

- The Outline pane on the left displays the structure of stylesheets you create.
- The Options pane in the middle lists stylesheets 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 stylesheets, such as name and description.



You can use the Report Explorer to work with stylesheets as follows.

Task	Pane to Use	Action
Create a stylesheet	Properties	Click the link that corresponds to the kind of stylesheet you want to create

Task	Pane to Use	Action
Open an existing stylesheet	Properties	Click the name of the stylesheet, which appear in the **Open Stylesheets** area
Select a stylesheet to use for converting an XML source file	Options	Select a stylesheet by clicking on it
View a list of customized styles in a stylesheet	Outline	Expand any open stylesheet
View a list of styles in a stylesheet	Outline or Options	Double-click the stylesheet
View a list of stylesheets 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)
View open stylesheets	Outline	Expand the Stylesheet Editor item in Report Explorer
Change the name or description of the current stylesheet	Properties	Edit the text in the **Display Name** or **Description** field.
Convert an XML source file using the current stylesheet	Properties	Click **Send to Source File Converter** in the Properties pane.
Edit customized style data	Properties	Click the style data item, which appears in the **Stylesheet Customizations** 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 stylesheet

Save a Stylesheet

You must save a stylesheet before you can use it to convert a source file or associate it with a report. To use the Report Explorer to save a stylesheet:

1 Select the stylesheet 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 stylesheet (to avoid overwriting built-in stylesheets). You must save the file in a folder in your MATLAB path for the stylesheet to appear in the Report Explorer. The file name must be unique in the MATLAB path.

   By convention, MATLAB Report Generator stylesheets have `.rgs` as their file name extension.

Delete a Stylesheet

To use the Report Explorer to delete a stylesheet that you created:

1 Select the stylesheet that you want to delete in the Outline pane on the left.

2 Click the stylesheet to delete from the Options pane in the middle.

3 Click **Delete stylesheet** in the stylesheet's Properties pane on the right.

You must restart the MATLAB software session for deleted stylesheets to disappear from the Options pane.

Note: You cannot delete built-in stylesheets.

Edit Stylesheet Data Items

In this section...
“Data Item Categories in Built-In Stylesheets” on page 8-9
“Edit Data Items in Simple or Advanced Edit Mode” on page 8-13
“Data Items” on page 8-13

Data Item Categories in Built-In Stylesheets

You can edit data items in built-in stylesheets 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 stylesheet provided with the MATLAB Report Generator software.

Categories of Styles in PDF (FO) Stylesheets

Category	Description of Data Items in Category
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

Category	Description of Data Items in Category
Pagination and General Styles	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 **Note:** 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 (`pi`), where 1 pica = 1/6 inch.
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
Stylesheet 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 stylesheets, see http://docbook.sourceforge.net/release/xsl/current/doc/ .

You can set up font mappings for non-English PDF fonts. The PDF stylesheets override those mappings. For details, see

Categories of Styles in HTML and Multi-Page HTML Stylesheets

Category of Style	Description of Data Items in Category
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

Category of Style	Description of Data Items in Category
Chunking	Options related to using an explicit TOC for chunking, depth of section chunks, navigational graphics, and display of titles in headers and footers
Stylesheet 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 stylesheet, 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 http://www.oasis-open.org/docbook/documentation/reference/html/docbook.html
- DocBook XSL stylesheets — see http://docbook.sourceforge.net/release/xsl/current/doc/

- DocBook print parameters, see http://docbook.sourceforge.net/release/dsssl/current/doc/print/

Categories of Styles in RTF (DSSSL) Stylesheets

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

Category of Style	Description of Data Items in Category
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 string that corresponds to the data in the stylesheet. This string 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 stylesheets in either simple edit mode or advanced edit mode. Edit values for RTF stylesheets in simple edit mode only. Data items in RTF stylesheets do not support advanced edit mode.

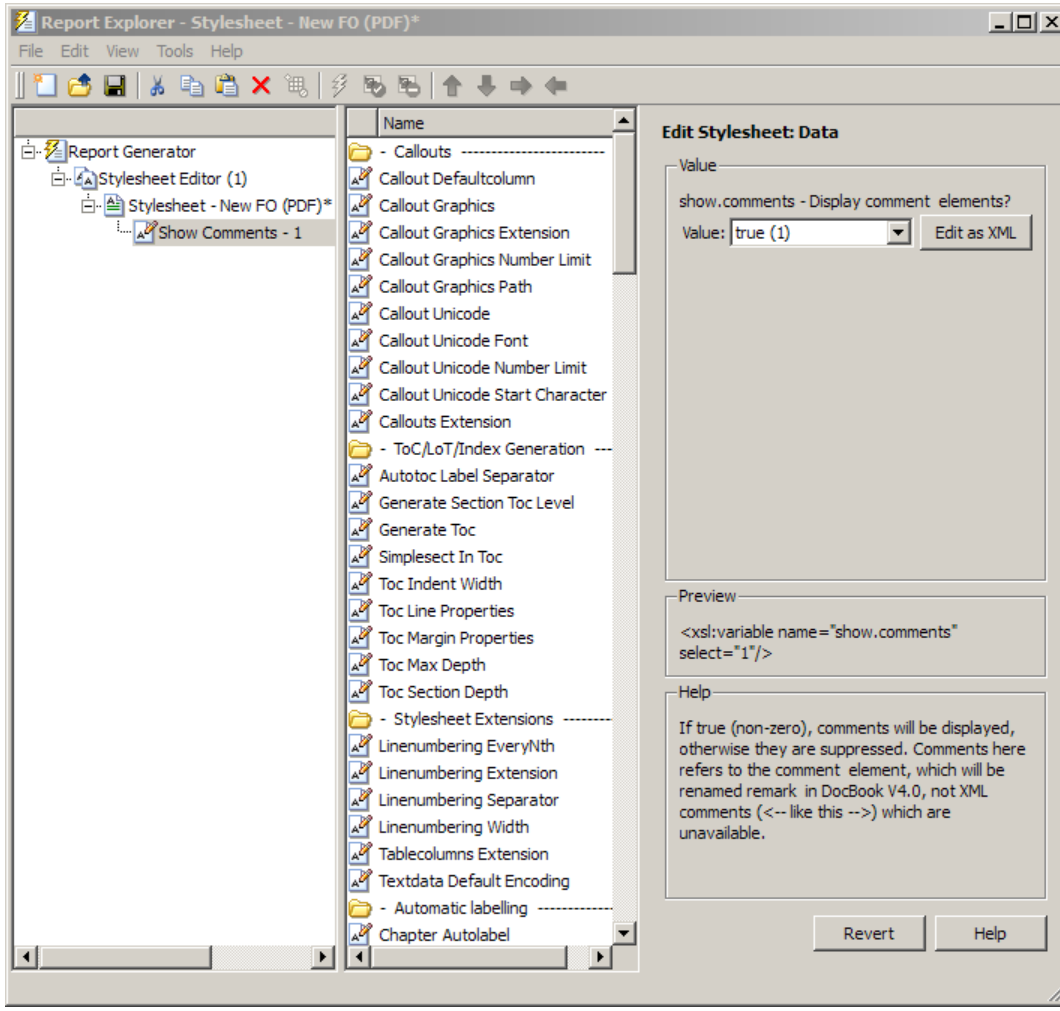
Note: To modify content for headers and footers you edit *stylesheet cells*, which do not appear in either simple or advanced mode. For more information, see "Stylesheet Cells for Headers and Footers" on page 8-23.

Data Items

Select a stylesheet 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

stylesheet. The Options pane in the middle shows a list of available stylesheet data items. The Properties pane on the right displays **Stylesheet Editor: 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 stylesheet 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 Strings

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

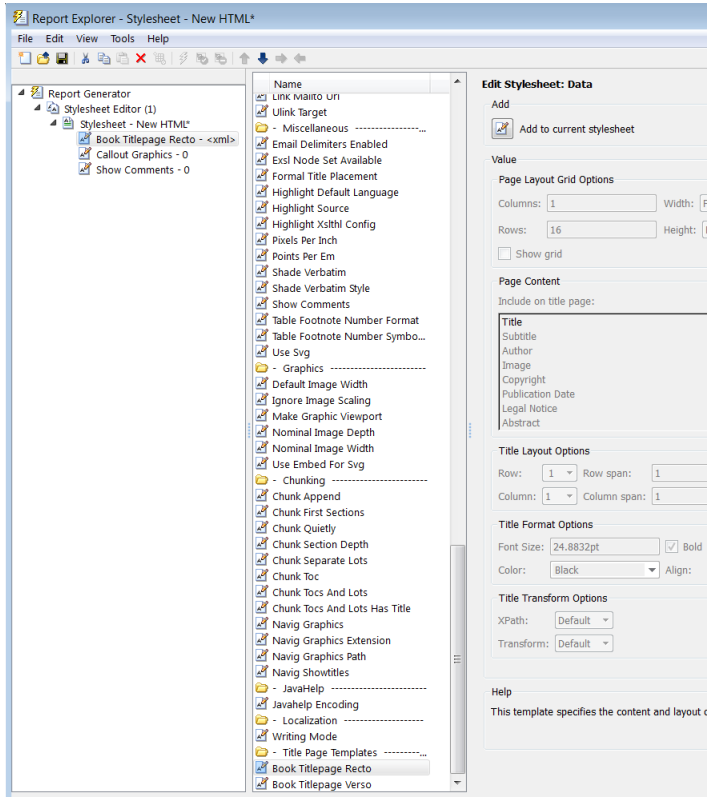
To make complex changes to a stylesheet, consider using Advanced edit mode. This enables you to edit XML expressions directly in the **Value (XML)** pane. If this pane does not appear, click **Edit as XML** to switch to advanced edit mode.

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 stylesheets, you can modify the layout, contents, and format of a title page by using the Stylesheet Editor.

1. In the Outline pane, select the stylesheet 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 stylesheet** 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.

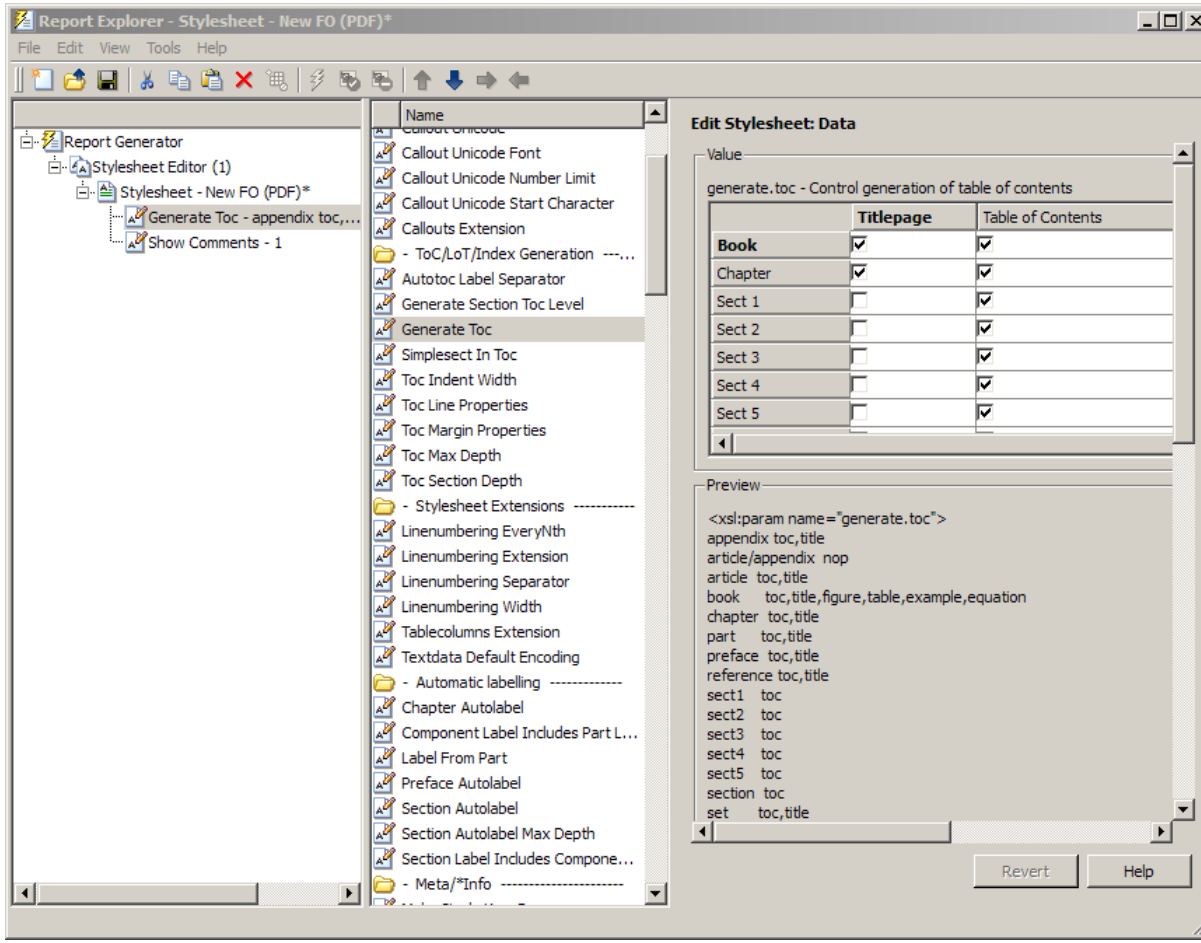
To specify properties for an individual title page element:

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** stylesheet. 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 8-20.

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

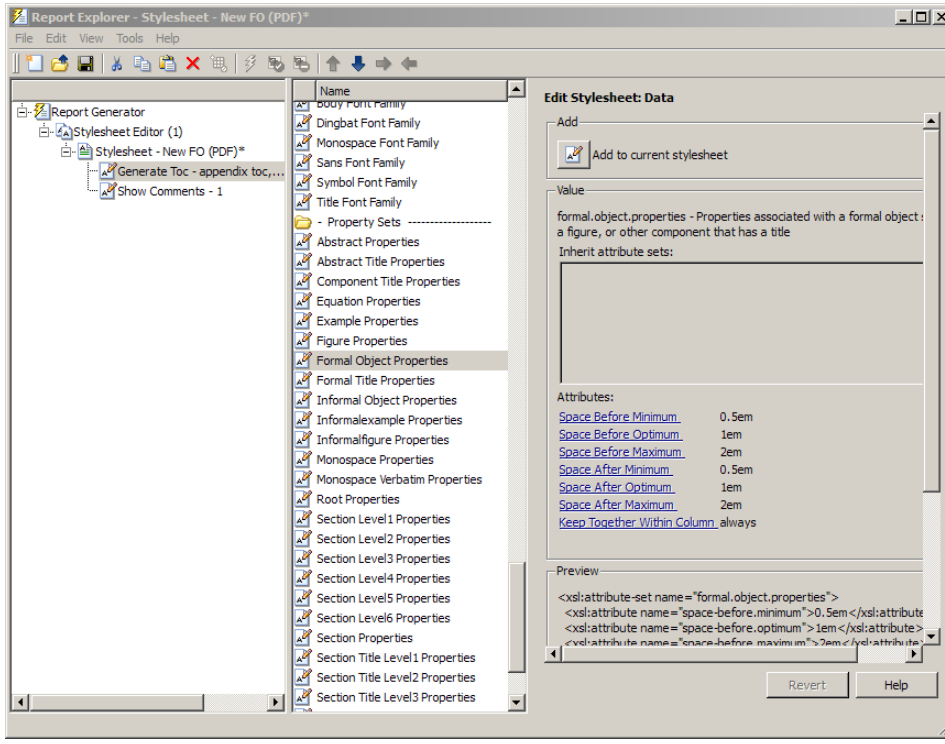
An *attribute set* consists of a group of attributes. Selecting an attribute set in the Outline pane on the left causes the Properties pane to list the attributes that belong to that set.

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 stylesheet 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 stylesheets appear as `varpair` data items, which are name/value pairs of information. RTF stylesheets are the only type of stylesheet that includes `varpair` data items.

Edit `varpair` data items as strings or as Boolean values. Boolean values appear as `true (#t)` and `false (#f)`.

Note: You cannot edit RTF stylesheet data items as XML.

Note: Data of type `varpair` is sometimes represented in stylesheets 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 stylesheets.

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**.

Stylesheet Cells for Headers and Footers

In this section...
“About Stylesheet Cells and Cell Groups” on page 8-23
“Headers and Footers” on page 8-24
“Add Content to Headers and Footers Using Templates” on page 8-26
“Insert Graphics Files” on page 8-27
“Modify Fonts and Other Properties” on page 8-27

About Stylesheet Cells and Cell Groups

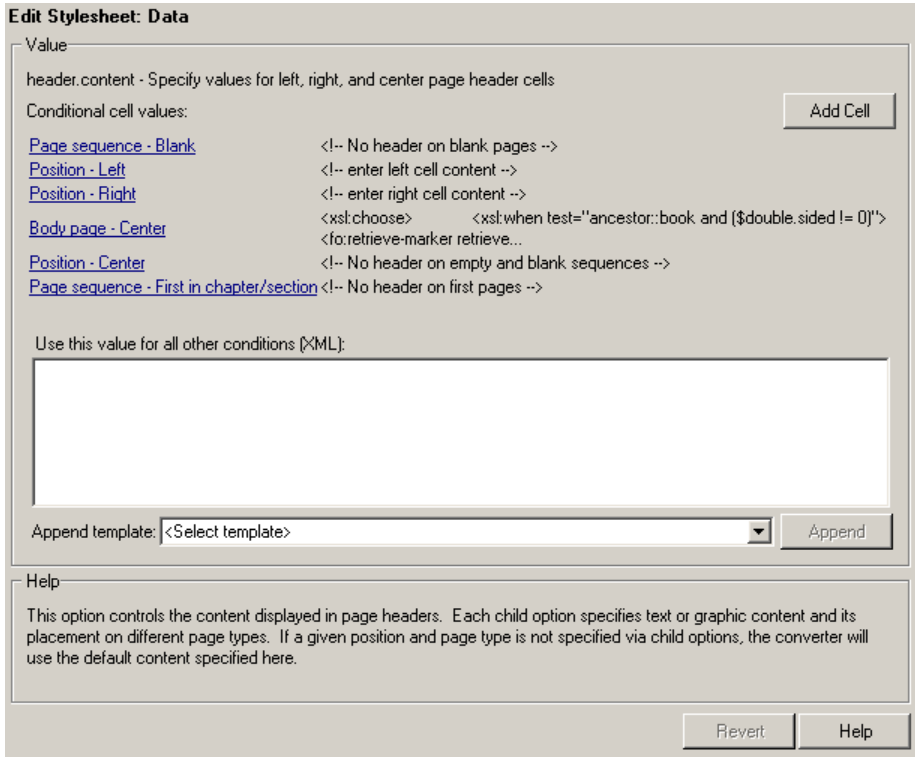
Use *stylesheet cells* to specify content of headers and footers in PDF reports.

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 stylesheet 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 stylesheet 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 stylesheet cell `Page sequence - Blank` specifies the content for a blank page; by default, the content is empty. Similarly, `Cell - Right Side` 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 stylesheet cell to specify content that satisfies specified conditions. The Properties pane for a stylesheet cell includes the following.

Label	Definition	Description
Condition	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 **Edit** to view or change a condition's XML code
Value (XML)	Content to appear in the report if the condition is met	Modify or create XML code for header or footer content
Append Template	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 8-26.

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 `http://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** stylesheet 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 8-30.

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 stylesheet 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 stylesheets. 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 8-35.

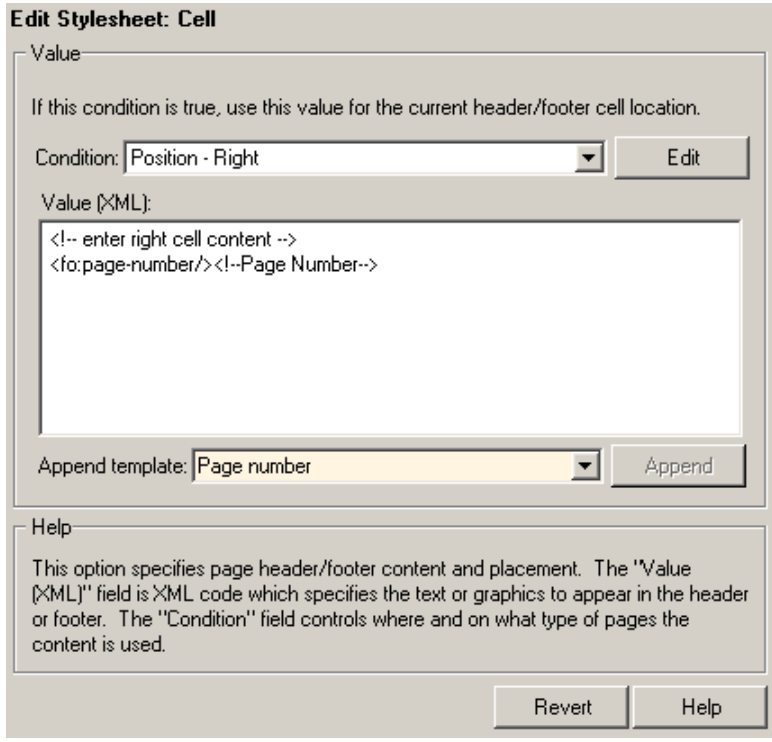
Customized Stylesheets

In this section...
“Number Pages in a Report” on page 8-29
“Add Graphics to Headers in PDF Reports” on page 8-30
“Change Font Size, Page Orientation, and Paper Type of a Generated Report” on page 8-35
“Edit Font Size as a Derived Value in XML” on page 8-37

Number Pages in a Report

This example shows how to edit a stylesheet cell to number the upper-right side of all pages in the generated report.

1. Define a basic stylesheet cell in the Header Content cell group with a condition of `right`.
   a. Open a PDF stylesheet 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 stylesheet. 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 stylesheet.

   a. Select **Tools** > **Edit Stylesheet** 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 stylesheet for PDF`.

   d. As **Description**, enter `Company logo in center of header`.

   e. Save the stylesheet as `logo_stylesheet.rgs` in a folder on your MATLAB path.

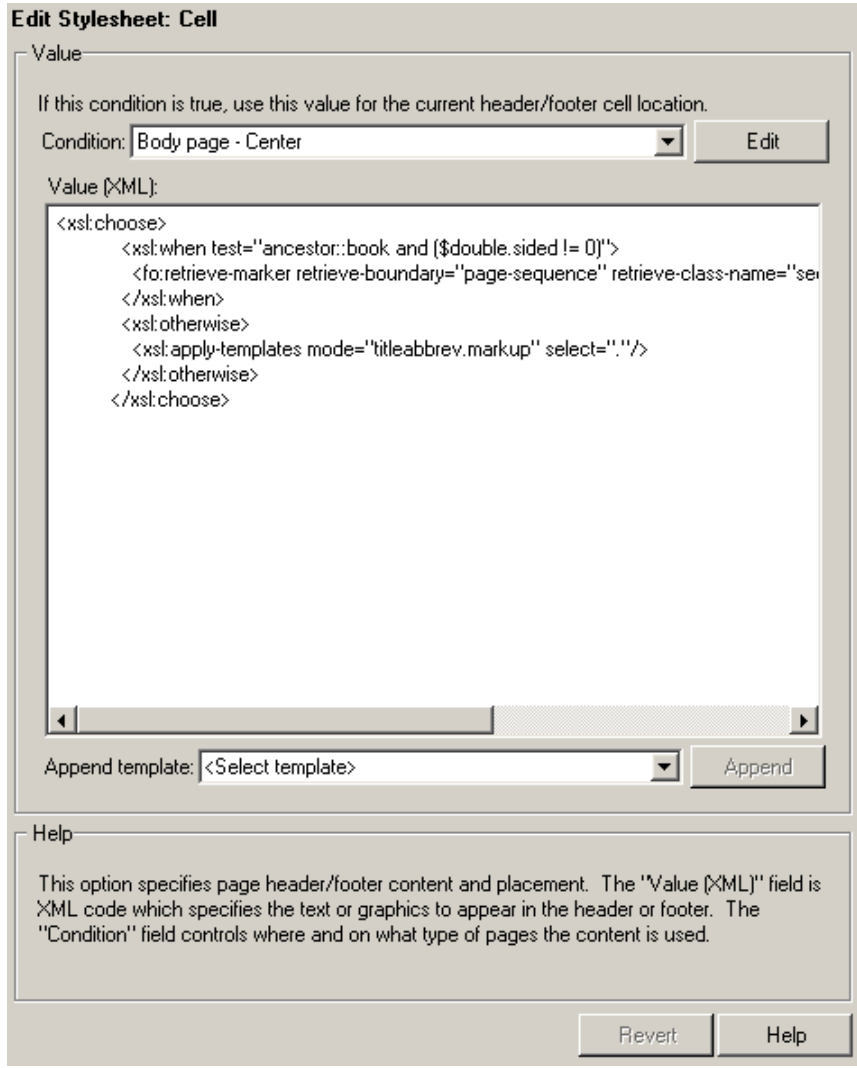
3. Open the cell group for editing.

   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.

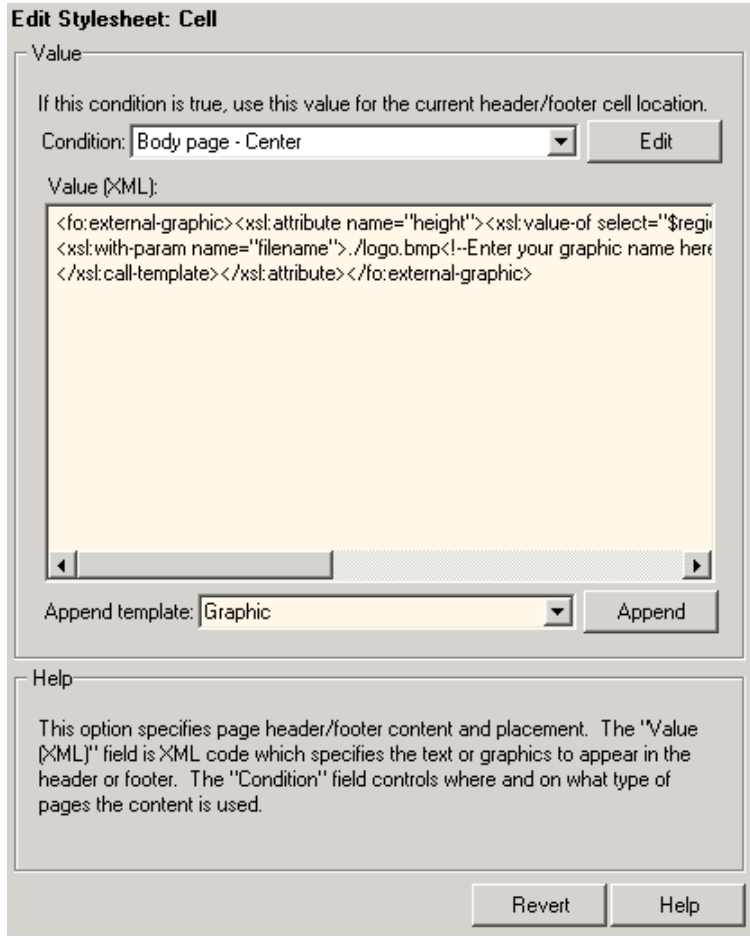
   The Properties pane appears as shown.



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.

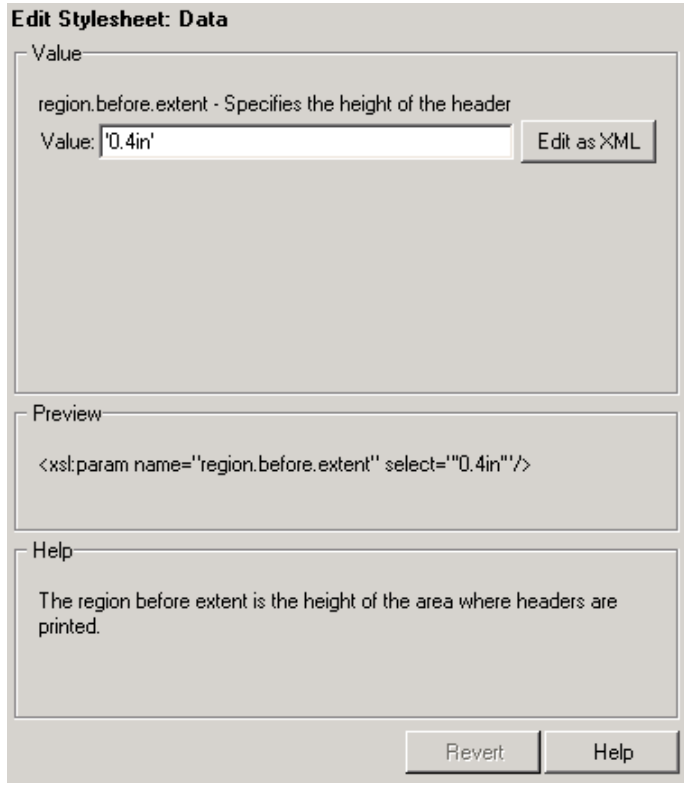


4 By default, the name of the graphic is `logo.bmp`. Change all instances of this name to `sample_logo.bmp` in the **Value (XML)** field.

5 Save the stylesheet.

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 `1.0in`.

c Save the stylesheet.

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 Stylesheet** area of the Properties pane on the right:

- Specify `Acrobat (PDF)` for **File format**
- Specify `Logo stylesheet 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 stylesheet by editing an existing stylesheet 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 Stylesheet** 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 **Stylesheet** 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** stylesheet.

b In the Properties pane on the right, enter `Custom Large Section Headers` as the stylesheet 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** stylesheet.

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 stylesheet**.

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` stylesheet.

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 a simple string, 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 stylesheet.

k Save the stylesheet as `customheader.rgs`, in a folder in your MATLAB path.

The `customheader.rgs` stylesheet appears as an available stylesheet 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 stylesheet to convert the current XML source file.

   a In the **Stylesheet Editor: Main** Properties pane on the right, click **Send to File Converter**

      The File Converter appears, with the `customheader.rgs` stylesheet 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 stylesheet.

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

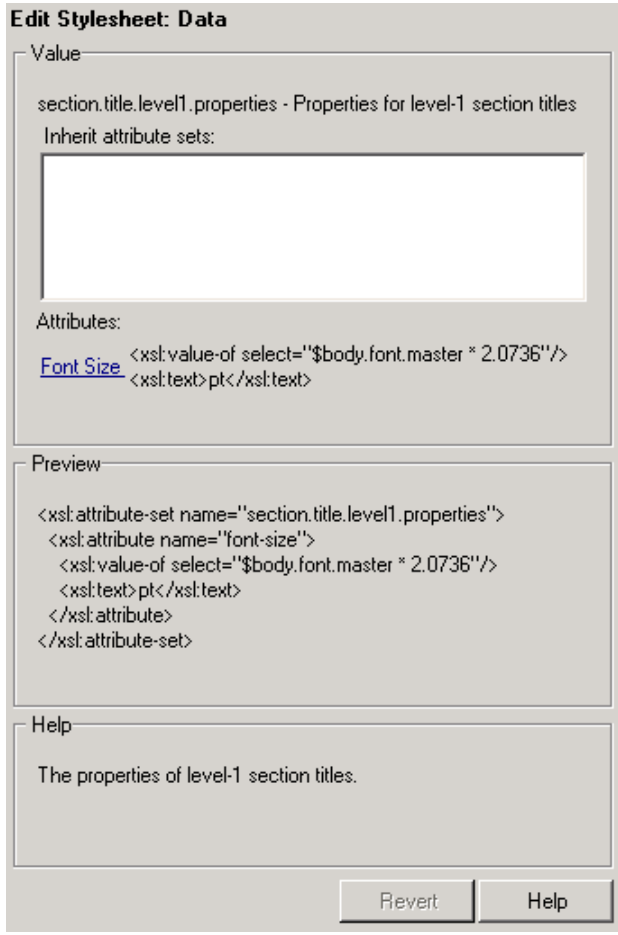
This example shows how to change the font size in a report to a value derived from other values. You do this by editing the PDF report's XML source directly.

1 Open the default print stylesheet for PDF documents.

2 In the Options pane in the middle, select and expand the **Property Sets** folder.

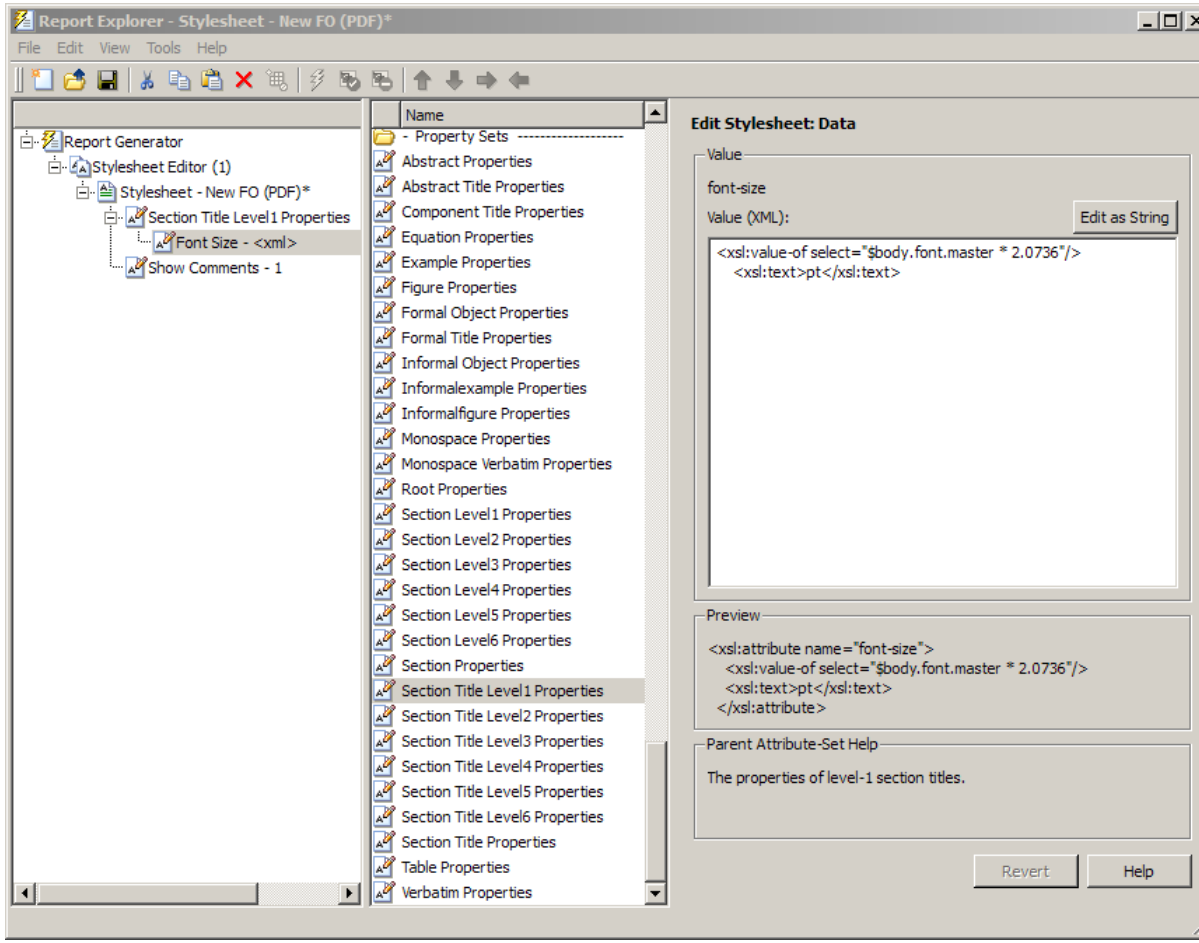
3 In the Options pane, double-click the **Section Title Level1 Properties** data item.

   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...
“Use Additional Fonts for PDF Output” on page 8-40
“PDF Font Support for Languages” on page 8-41
“Identifying When to Specify a Font” on page 8-42
“Stylesheets Override PDF Font Mapping” on page 8-42
“Non-English PDF Font Mapping Tasks” on page 8-42
“lang_font_map.xml File” on page 8-42
“Locate Non-English Fonts” on page 8-44
“Add or Modify Language Font Mappings” on page 8-46
“Specify the Location of Font Files” on page 8-46

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 further configure your setup to:

- Use additional fonts. See “Use Additional Fonts for PDF Output” on page 8-40.
- Map to a different font for your locale. See “PDF Font Support for Languages” on page 8-41.

Use Additional Fonts for PDF Output

During report generation, the Report Generator uses font descriptions to create the text in your report. If you want your report program to use fonts other than the defaults used by the Report Generator for PDF output, copy the font descriptions to a specific folder in your file system before you generate your report.

By default, the Report Generator looks for additional font descriptions in your personal `Temp` folder, in the folder `rptgenFonts`. You can specify a different name for the folder. You can alternatively configure the Report Generator to look in a folder in the MATLAB installation path.

To change the location for the fonts, edit the configuration file `fop_config.xml`. Go to the folder where the file is located using this MATLAB command:

```
cd(fullfile(matlabroot,'toolbox','shared','rptgen','@rpt_xml','@db_output'))
```

Open the file `fop_config.xml` in a text editor. Set the location in the `<directory>` tag. The configuration file includes the instructions for editing the location.

Copy Font Descriptions During Report Generation

This procedure assumes that you are using the default folder for font descriptions `$tempdir/rptgenFonts`.

1. Copy the font descriptions you need to a convenient folder in your file system.
2. Add code to your report program that copies your font descriptions to the folder specified in the FOP configuration file. This font description folder is deleted after report generation.

   Set `myfontdir` to the location of your font descriptions.

   ```
   myfontdir = 'x:\rptgenFonts';
   copyfile(myfontdir,fullfile(tempdir,'rptgenFonts'));
   ```

   In Report Explorer, add the code in an `Eval` component. For a DOM API report program, add the code before the program writes the report.

PDF Font Support for Languages

When generating PDF reports, the MATLAB Report Generator uses a font capable of rendering text in these languages:

- English
- Japanese
- Korean
- Russian (Cyrillic)

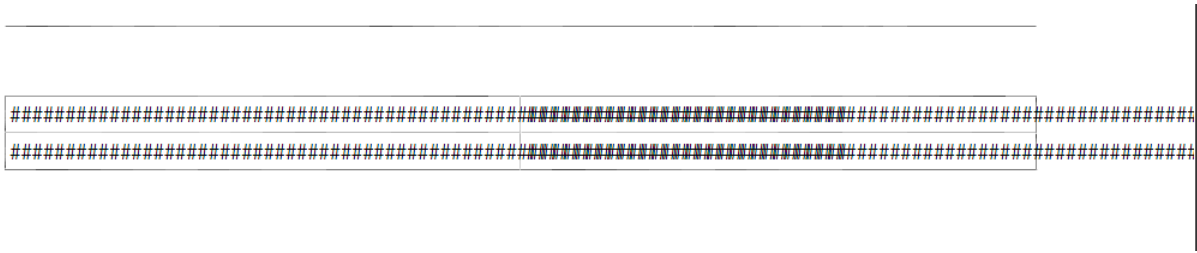
The Report Generator uses a font map to determine the font appropriate for a particular locale. The font map specifies a default set of fonts. You can modify the map to:

- 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:



Stylesheets Override PDF Font Mapping

PDF stylesheets for the MATLAB Report Generator specify fonts for body text, copyright, quotes, symbols, dingbats, monospace, sans serif, and titles.

The PDF stylesheet settings override the PDF font mapping entries.

If you do not specify a PDF stylesheet, 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 8-44
- “Add or Modify Language Font Mappings” on page 8-46
- “Specify the Location of Font Files” on page 8-46

lang_font_map.xml File

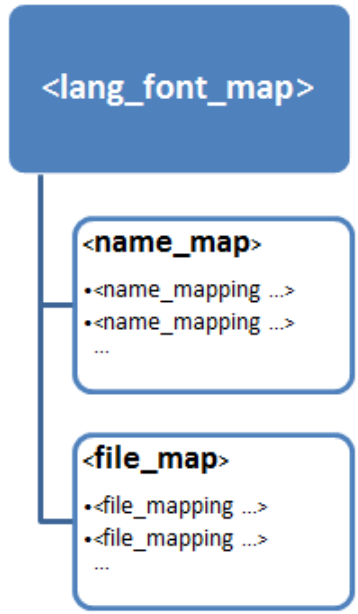
Use an XML editor with the `lang_font_map.xml` file to enter all the PDF font mappings for your reports.

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="glnx" usage="body">FreeSerif, Regular</name_mapping>
    <name_mapping lang="en" platform="glnx" usage="monospace">FreeMono, Regular</name_mapping>
    <name_mapping lang="en" platform="glnx" usage="sans">FreeSans, Regular</name_mapping>
    <name_mapping lang="en" platform="glnx" 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="glnx" name="FreeSerif, Regular">FreeSerif.ttf</file_mapping>
    <file_mapping lang="en" platform="glnx" name="FreeMono, Regular">FreeMono.ttf</file_mapping>
    <file_mapping lang="en" platform="glnx" name="FreeSans, Regular">FreeSans.ttf</file_mapping>
    <file_mapping lang="en" platform="glnx" 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 stylesheet.

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 *`winder`* 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.

9

Comparing XML Files

- "Compare XML Files" on page 9-2
- "How to Compare XML Files" on page 9-4
- "Explore the XML Comparison Report" on page 9-6
- "How the Matching Algorithm Works" on page 9-10

Compare XML Files

You can use MATLAB Report Generator software to compare a pair of XML text files.

The XML comparison tool processes the results into a report that you can use to explore the file differences.

You can access the XML comparison tool from:

- The MATLAB Current Folder browser context menu
- The MATLAB command line.

The XML comparison tool compares the files using the "Chawathe" algorithm, as described in this paper:
Change Detection in Hierarchically Structured Information, Sudarshan Chawathe, Anand Rajaraman, and Jennifer Widom; SIGMOD Conference, Montreal, Canada, June 1996, pp. 493-504.

This conference paper is based upon work published in 1995: see `http://dbpubs.stanford.edu:8090/pub/1995-45`.

XML comparison reports display in the Comparison Tool. For more information about the Comparison Tool, see "Comparing Files and Folders" in the MATLAB documentation.

The XML comparison report shows a hierarchical view of the portions of the two XML files that differ. The report does not show sections of the files that are identical.

If the files are identical you see a message reporting there are no differences.

Note: It might not be possible for the analysis to detect matches between previously corresponding sections of files that have diverged too much.

Change detection in the Chawathe analysis is based on a scoring algorithm. Items match if their Chawathe score is above a threshold. The MATLAB Report Generator implementation of Chawathe's algorithm uses a comparison pattern that defines the thresholds. For more information, see "How the Matching Algorithm Works" on page 9-10.

For information about creating and using XML comparison reports, see:

1 “How to Compare XML Files” on page 9-4
2 “Explore the XML Comparison Report” on page 9-6

How to Compare XML Files

In this section...
“Select Files to Compare” on page 9-4
“Change Comparison Type” on page 9-5
“XML Comparison Examples” on page 9-5
“See Also” on page 9-5

Select Files to Compare

- “From the Current Folder Browser” on page 9-4
- “From the Comparison Tool” on page 9-5
- “From the Command Line” on page 9-5

From the Current Folder Browser

To compare two files from the Current Folder browser:

- For two files in the same folder, select the files, right-click and select **Compare Selected Files/Folders**.
- To compare files in different folders:

  1. Select a file, right-click and select **Compare Against**
  2. Select the second file to compare in the Select Files or Folders for Comparison dialog box.
  3. Leave the default **Comparison type**, `XML text comparison`.
  4. Click **Compare**.

If the selected files are XML files, the XML comparison tool performs a Chawathe analysis on the files and displays a report in the Comparison Tool.

The file you right-click to launch the XML comparison tool displays on the right side of the report.

For more information about comparisons of other file types with the Comparison Tool, such as text, MAT or binary, see “Comparing Files and Folders” in the MATLAB documentation.

From the Comparison Tool

To compare files using the Comparison Tool, from the MATLAB Toolstrip, in the **File** section, select the **Compare** button. In the dialog box select files to compare.

If the files you select to compare are XML files and you select an `XML comparison`, the XML comparison tool performs a Chawathe analysis of the XML files, and generates an report.

From the Command Line

To compare XML files from the command line, enter

```
visdiff(filename1, filename2)
```

where `filename1` and `filename1` are XML files. This XML comparison functionality is an extension to the MATLAB `visdiff` function.

Change Comparison Type

If you specify two XML files to compare using either the Current Folder Browser or the `visdiff` function, then the Comparison Tool automatically performs the default comparison type, `XML text comparison`.

To change comparison type, either create a new comparison from the Comparison Tool, or use the **Compare Against** option from the Current Folder browser. You can change comparison type in the Select Files or Folders for Comparison dialog box. If you want the MATLAB text differences report for XML files, change the comparison type to `Text comparison` in the dialog before clicking **Compare**.

XML Comparison Examples

For an example with instructions, see `mlxml_testplan`.

See Also

For an overview, see “Compare XML Files” on page 9-2.

For explanations on how to use and understand the report and the XML comparison functionality, see “Explore the XML Comparison Report” on page 9-6

Explore the XML Comparison Report

In this section...
"Navigate the XML Comparison Report" on page 9-6
"Save Comparison Log Files in a Zip File" on page 9-8
"Export Results to the Workspace" on page 9-8

Navigate the XML Comparison Report

The XML comparison report shows changes only. The report is a hierarchical view of the differences between two XML text files, and is not a hierarchical view of the original XML data.

To *step through differences*, use the **Comparison** tab on the toolstrip. To move to the next or previous group of differences, on the **Comparison** tab, in the **Navigate** section, click the arrow buttons to go to the previous or next difference.

You can also click to select items in the hierarchical trees.

- Selected items appear highlighted in a box.
- If the selected item is part of a matched pair it is highlighted in a box in both left and right trees.

Report item highlighting indicates the nature of each difference as follows:

Type of report item	Highlighting	Notes
Modified	Pink	Modified items are matched pairs that differ between the two files. When you select a modified item it is highlighted in a box in both trees. Changed parameters for the selected pair are displayed in a separate **Parameters** panel for review. If strings are too long to display in the **Parameters** table, right-click and select **Compare as Text** to open a new comparison of the parameters.
Unmatched	Green	When you select an unmatched item it is highlighted in a box in one tree only.

Type of report item	Highlighting	Notes
Container	None	Rows with no highlighting indicate a container item that contains other modified or unmatched items.

Use the toolbar buttons or the **Comparison** menu for the following functions:

- **Refresh** — Run Chawathe analysis again to refresh the comparison report.
- **Swap Sides** — Swap sides and rerun comparison. Runs the Chawathe analysis again.
- **Save As > Save as HTML** — Opens the Save dialog box, where you can choose to save a printable version of the XML comparison report. The report is a noninteractive HTML document of the differences detected by the Chawathe algorithm for printing or archiving a record of the comparison.
- **Save As > Save to Workspace** — Export XML comparison results to workspace.
- In the **Navigate** section, click the arrow buttons (or press Up or Down keys) to go to the previous difference or go to the next difference.
- **Compare Selected Parameter** — Open a new report for the currently selected pair of parameters. Use this when the report cannot display all the details in the Parameters pane, e.g., long strings or a script.

Use the **View** tab controls on the toolstrip for the following functions:

- **Expand All** — Expands every item in the tree.

  Tip Right-click to expand or collapse the hierarchy within the selected tree node.

- **Collapse All** — Collapses all items in the tree to the most compact view possible.

See also "XML Comparison Examples" on page 9-5.

To understand the results within an XML comparison report, try reading the documentation section on "How the Matching Algorithm Works" on page 9-10.

Note: It may not be possible for the analysis to detect matches between previously corresponding sections of files that have diverged too much.

Save Comparison Log Files in a Zip File

Temporary XML comparison files accumulate in *tempdir*`/MatlabComparisons/XMLComparisons/TempDirs/`. These temporary files are deleted when you close the related comparison report.

You can zip the temporary files (such as log files) created during XML comparisons, for sharing or archiving. While the comparison report is open, enter:

```
xmlcomp.zipTempFiles('c:\work\myexportfolder')
```

The destination folder must exist. The output reports the zip file name:

```
Created the zipfile "c:\work\myexportfolder\20080915T065514w.zip"
```

To view the log file for the last comparison in the MATLAB Editor, enter:

```
xmlcomp.showLogFile
```

Export Results to the Workspace

To export the XML comparison results to the MATLAB base workspace,

1. On the **Comparison** tab, in the **Comparison** section, select **Save As** > **Save to Workspace**.

   The Input Variable Name dialog box appears.
2. Specify a name for the export object in the dialog and click **OK**. This action exports the results of the XML comparison to an `xmlcomp.Edits` object in the workspace.

The `xmlcomp.Edits` object contains information about the XML comparison including file names, filters applied, and hierarchical nodes that differ between the two XML files.

To create an `xmlcomp.Edits` object at the command line without opening the Comparison Tool, enter:

```
Edits = xmlcomp.compare(a.xml,b.xml)
```

Property of `xmlcomp.Edits`	Description
`Filters`	Array of filter structure arrays. Each structure has two fields, Name and Value.

Property of `xmlcomp.Edits`	Description
`LeftFileName`	File name of left file exported to XML.
`LeftRoot`	`xmlcomp.Node` object that references the root of the left tree.
`RightFileName`	File name of right file exported to XML.
`RightRoot`	`xmlcomp.Node` object that references the root of the right tree.
`TimeSaved`	Time when results exported to the workspace.
`Version`	MathWorks® release-specific version number of `xmlcomp.Edits` object.

Property of `xmlcomp.Node`	Description
`Children`	Array of `xmlcomp.Node` references to child nodes, if any.
`Edited`	Boolean — If `Edited = true` then the node is either inserted (green) or part of a modified matched pair (pink).
`Name`	Name of node.
`Parameters`	Array of parameter structure arrays. Each structure has two fields, Name and Value.
`Parent`	`xmlcomp.Node` reference to parent node, if any.
`Partner`	If matched, `Partner` is an `xmlcomp.Node` reference to the matched partner node in the other tree. Otherwise empty `[]`.

How the Matching Algorithm Works

In this section...
"How the Chawathe Algorithm Works" on page 9-10
"Why Use a Heuristic Algorithm?" on page 9-12
"Examples of Matching Results" on page 9-12

How the Chawathe Algorithm Works

The XML comparison tool compares the XML files using the "Chawathe" algorithm, as described in the paper:
Change Detection in Hierarchically Structured Information, Sudarshan Chawathe, Anand Rajaraman, and Jennifer Widom; SIGMOD Conference, Montreal, Canada, June 1996, pp. 493-504.

The core of the XML file comparison engine is Chawathe's matching algorithm. This matching algorithm is a heuristic method based on a scoring system.

XML text documents are hierarchical data structures. Users can insert, delete, or reorder elements, modify their contents, or move elements across different parts of the hierarchy. The Chawathe algorithm can detect these different types of changes within the hierarchy of the document. As with conventional text differencing utilities, the Chawathe algorithm detects local text that is added, deleted, or changed, and additionally can prepare an edit script that can be used to create a report of the hierarchical location of detected differences.

The Chawathe algorithm attempts to match elements that are of the same category. The Chawathe paper refers to these categories as *labels*. In the following XML example documents (with labels `A`, `B`, and `C`):

- The three `C` elements on the left are compared with the three `C` elements on the right
- The single `B` element on the left is compared with the two `B` elements on the right

<A> <C> First </C> <C> Second </C> </A> <B Name="first"> Some text </B> <A> <C> Third </C> </A>	<A> <C> First </C> <C> Third </C> <B Name="second"> More text </B> </A> <B Name="first"> Modified text </B> <A> <C> Fourth </C> </A>

The Chawathe algorithm matches a particular label by extracting a flat sequence of elements from the hierarchical document and attempting to match the elements in the sequences. In the example above, elements of the sequence

```
(<C> First </C>, <C> Second </C>, <C> Third </C>)
```

are matched against elements of the sequence

```
(<C> First </C>, <C> Third </C>, <C> Fourth </C>)
```

Sequences are matched using a Longest Common Subsequence (LCS) algorithm. For example, if `C` elements are matched on their text content, the LCS of the above sequences is given by:

```
(<C> First </C>, <C> Third </C>)
```

You can define a *score* for matching elements of a particular label in different ways. For instance, in the above example, `C` elements can be matched on text content, `B` can be matched on text content and on `Name`, and `A` on the number of `B` and `C` elements they have in common. To determine whether elements match or not, the Chawathe algorithm compares the score to a threshold.

The implementation can specify scoring methods, thresholds, the definition of labels, and the order in which labels are processed. These can be defined separately for each problem domain or type of XML file. The XML comparison tool provides suitable definitions for a set of common XML file types, and uses a default definition for any type of XML document it does not recognize.

Why Use a Heuristic Algorithm?

Chawathe's algorithm is a heuristic. That is, it cannot guarantee to return the optimal matching between two sequences. It is the use of a threshold mechanism in combination with an LCS algorithm that makes the algorithm a heuristic. A heuristic algorithm is preferable to an optimal matching because the heuristic is much faster.

An algorithm can only guarantee a mathematically optimal matching by exhaustively computing the score between all pairs of elements in the two sequences and choosing those pairs that maximize an overall matching score between the two sequences. This exhaustive approach is computationally very expensive because its running time increases exponentially with the length of the sequences to be matched.

Also a user's expectations can depend on context information that is not available to the matching algorithm (e.g., prior knowledge of the precise sequence of changes applied). This means even a mathematically optimal algorithm might match elements contrary to a user's expectations.

In contrast with the mathematically optimal approach, Chawathe's algorithm guarantees linear running time for sequences that are the same or very similar. The worst-case scenario is quadratic running time for sequences that are entirely different.

The XML comparison tool performs best when the files to be compared are mostly similar. It becomes slower for files that contain more differences.

Examples of Matching Results

- "Elements Matched in Previous Comparisons Fail to Match" on page 9-12
- "Elements Matched Across Different Parts of the Hierarchy" on page 9-13
- "Two Sequences of Elements Are Cross-Matched" on page 9-14

Elements Matched in Previous Comparisons Fail to Match

Elements could fail to match even if they were matched in comparisons of previous versions of the documents. A seemingly small change in one of the properties used for matching can cause this to happen if it tips the score under the threshold.

Consider the following example where

- `B` elements are scored on the value of `x`

- A elements are scored on the ratio of matching B elements
- For both A and B the score is compared with a threshold of 0.5.

<A> <B x="1"/> <B x="2"/> <B x="3"/> </A>	<A> <B x="1"/> <B x="7"/> <B x="3"/> </A>	<A> <B x="1"/> <B x="7"/> <B x="6"/> </A>

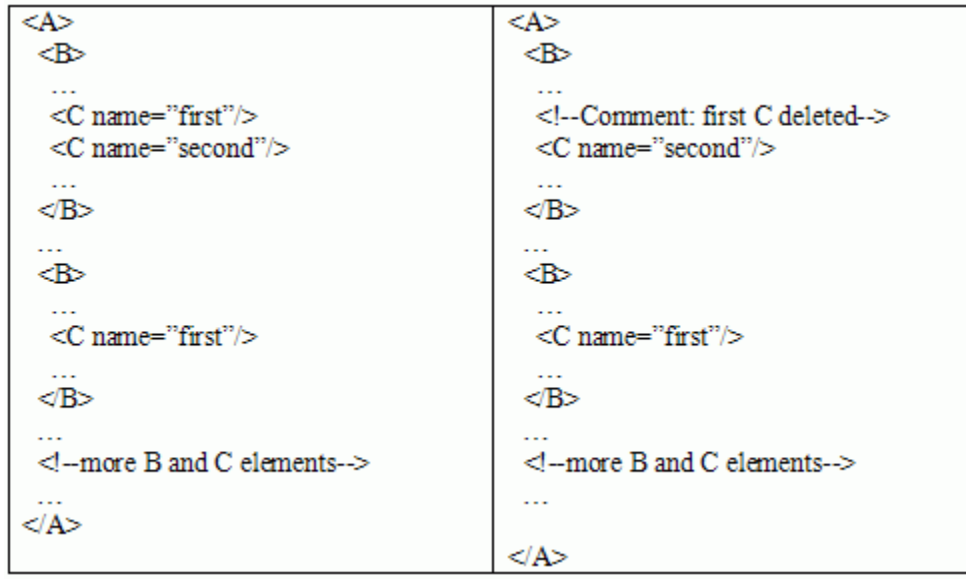
The left A and the middle A have two out of three B elements in common, resulting in a matching score of 2/3=0.66. The XML comparison tool marks the A elements as matched and the report shows that their contents have been *modified*.

When a user makes a further change to the middle document (resulting in the right document), and this new document is compared again to the left document, the matching score for A drops to 1/3=0.33. The algorithm considers the A elements unmatched this time. In this case, the difference between the two documents is marked as a *deletion* of A from the left document and an *insertion* of a new A into the right document.

This problem is likely to occur when there is little information available inside a single element to score a match. A seemingly small change in one of the properties used for matching could tip the score under the threshold, and therefore result in a large change in the outcome of the comparison.

Elements Matched Across Different Parts of the Hierarchy

Sometimes matches of similar items occur across different parts of the hierarchy. In the following example, C elements are matched on `name`:



In this case, the user might expect to see the very *first* `C` element on the left marked as deleted, with the second and third `C` elements matched to the corresponding `C` element on the right. However, this might not happen, if the first `C` on the left is matched to the second `C` on the right, even though these two `C` elements exist in very different parts of the document hierarchy. This mismatch would result in the third `C` element on the left being marked as deleted.

This case is likely to occur when there are several potential matching candidates for a particular element. In other words, when elements of a particular label tend to be very similar. Whether such a spurious cross-matching occurs or not depends on all of the other `C` elements within the two documents. The LCS algorithm used for matching the two sequences favors *local* matches over *distant* ones. In other words, sub-sequences of elements that are close together in the first sequence tend to be matched to sub-sequences of elements that are close together in the second sequence. However, this locality is not always guaranteed, and the outcome depends on how other elements in the sequence are matched.

Two Sequences of Elements Are Cross-Matched

It is difficult to distinguish many similar potential matches. In the following example, `B` elements are scored on `name`, `p1`, and `p2`, and the score is compared to a threshold of 0.5.

<A> <B name="1" p1="false" p2="on"/> <B name="2" p1="false" p2="on"/> <B name="3" p1="false" p2="on"/> <B name="4" p1="false" p2="off"/> </A> … <!--more A and B elements--> …	<A> <B name="1" p1="false" p2="on"/> <B name="2" p1="false" p2="on"/> <B name="new" p1="false" p2="on"/> <B name="3" p1="false" p2="on"/> <B name="4" p1="false" p2="on"/> </A> … <!--more A and B elements--> …

The right document contains one B element more than the left document, and therefore one of the B elements on the right must remain unmatched and the tool will mark one as inserted. However, since most B elements on the left potentially match most B elements on the right, it is impossible to predict exactly how the sequences will be matched. For instance, the comparison could generate the following result:

```
“B name=”1” …” > “B name=”2” …”
“B name=”2” …” > “B name=”new” …”
“B name=”3” …” > “B name=”3” …”
“B name=”4” …” > “B name=”4” …”
```

In this case, “B name= “1” on the right remains unmatched. As in the previous example, this depends on how all of the other B elements in the two documents are matched. This situation is likely to occur when elements have several potential matching candidates.

Components — Alphabetical List

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 5-21.

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**: Creates a hyperlink to the object in the generated report.

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`
  - `PNG 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` in the MATLAB documentation.
  - `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

By default, chapters are numbered and sections are not numbered. Specify chapter and section numbering using a stylesheet. For more information about chapter and section numbering options in Web and print stylesheets, see “Report Output Format” on page 4-4.

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/Section 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 example, if you use a Word template that defines a `Heading 2` style, you can enter `Heading 2` as the style name. For more information about template styles, see "Report Conversion Templates" on page 6-2.

- **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`

Related Examples

- "Customize Microsoft Word Part Templates"

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 4-18, `Import File`, `Nest Setup File`, `Stop Report Generation`, `Time/Date Stamp`

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”.
- **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 5-21.

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**: Creates a hyperlink to the object in the generated report.

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`
  - `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` in the MATLAB documentation.

- **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**: Creates a hyperlink to the object in the generated report.

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 string 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 5-6.

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 5-6.

- **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. For more information about this notation, see "%<VariableName> Notation" on page 10-102 on the `Text` component reference page.
- **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 of a given text string 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

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
    ```

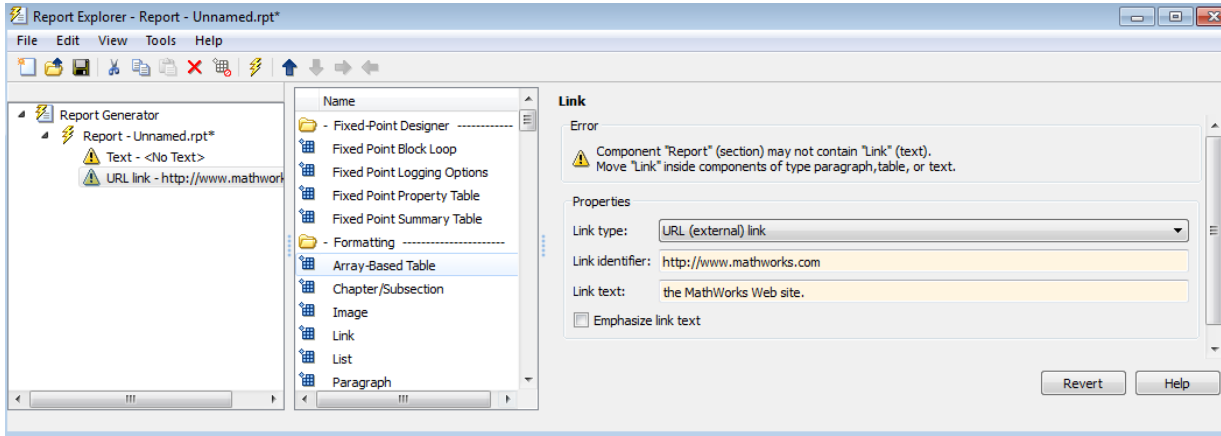
  For a link to a MATLAB command, enter `matlab:` followed by a space and the MATLAB command that you want the link to execute.

- **Link text**: Specifies text to use in the link.
- **Emphasize link text**: Italicizes the link text.

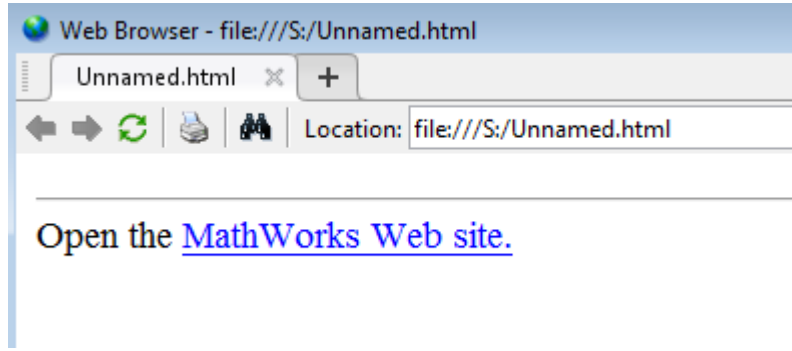
Examples

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 string).
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 `http://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`.

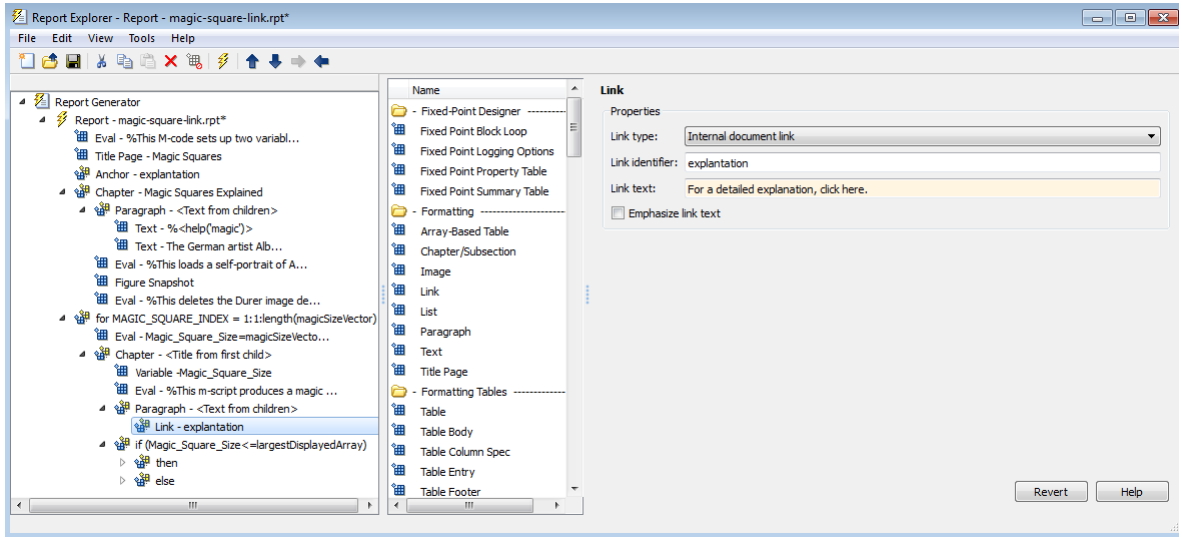
5 In the Outline pane, under the second Chapter component, select the Eval component.

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.

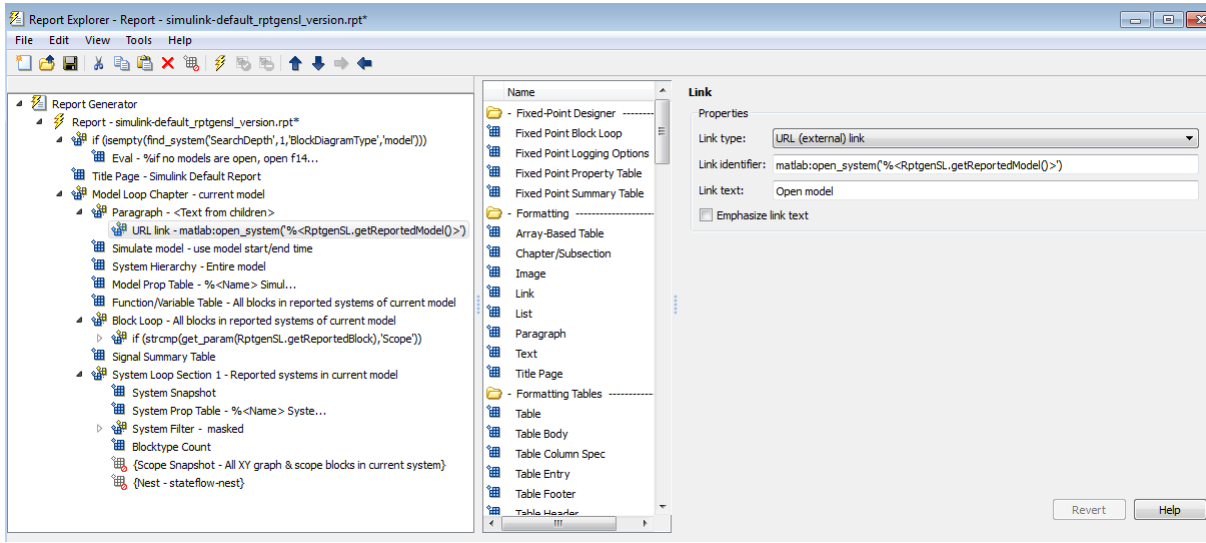
64	2	3	61	60	6	7	57
9	55	54	12	13	51	50	16
17	47	46	20	21	43	42	24
40	26	27	37	36	30	31	33
32	34	35	29	28	38	39	25
41	23	22	44	45	19	18	48
49	15	14	52	53	11	10	56
8	58	59	5	4	62	63	1

10 Click the link to move to near the top of the report, to “Chapter 1. Magic Squares Explained.”.

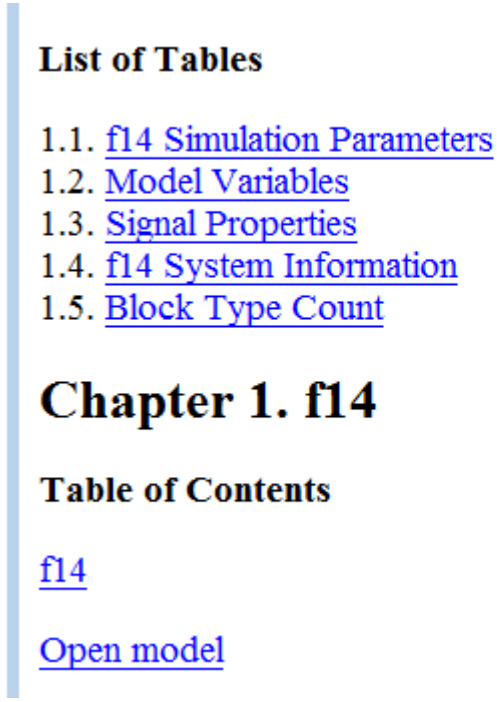
Link to a Model

This example shows how to add a link to a Simulink model. To view the model, you must have the Simulink software installed.

1. Open Report Explorer with the `setedit` command.
2. In the middle pane, click `simulink-default_rptgensl_version.rpt`.
3. In the Library pane in the middle, under the Formatting category, select the Paragraph component and click the **Add component** icon.
4. In the Library pane in the middle, under the Formatting category, select the Link component and click the **Add component** icon.
5. In the Properties pane:
   - Set **Link type** to `URL (external) link`.
   - In **Link Identifier**, enter `matlab:open_system('%<RptgenSL.getReportedModel()>')`.
   - In **Link text**, enter `Open model`.



6. Generate the report.



7 Click the “Open model” link to open the model.

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 Conversion Templates" on page 6-2.

List Formatting

- **List style**:
  - `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 a specified string 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 5-6.

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 Simulink Manifest Tools analysis to determine what products appear in the version information table. See "Analysis Limitations" for Manifest Tools analysis limitations.

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                [-] Report - R2.rpt
 [ ] Chapter                              [ ] 1
      [-] B                               [ ] 2
         [ ] Nest Setfile - R2.rpt       [-] Chapter
         [ ] C                              [ ] 4
      [ ] D                                 [ ] 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

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 the style to use with the paragraph 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 Paragraph properties dialog box, set **Title Options** to one of these values:
   - `Get title from first child`
   - `Custom title`
3. Set **Style Name** to `Specify`.
4. In the **Style Name** text box, type a style name.

   To take effect, the style you specify 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 Conversion Templates" on page 6-2.

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. Suppose that you then apply a text style named `BoldStyle` to text in the paragraph. The text formatted with `BoldStyle` toggles the paragraph style bold off.

One way to make sure that the text is bold is to 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, nothing appears in the report.

Use the `%<VariableName>` notation in this field if you want to insert the value of a variable from the MATLAB workspace. For more details about this notation, see "%<VariableName> Notation" on page 10-102 on the `Text` component reference page.

Style Name

Specifies the style to use with the paragraph 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 style (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 Conversion Templates" on page 6-2.

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.
- **Retain spaces and carriage returns**: Formats text in the report in the same way as it is entered.
- **Show text as syntax-highlighted MATLAB code**: Displays the text as syntax-highlighted MATLAB code.
- **Color**: Specifies the color of the text.
  - 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.

Insert Anything into Report?

Yes, depending on child components.

Class

`rptgen.cfr_paragraph`

See Also

Chapter/Subsection, Empty Component, Image, Link, List, Table, Text, Title Page

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`

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 Conversion Templates" on page 6-2.
- **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 Conversion Templates" on page 6-2.

- **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 (`pi`), 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 Conversion Templates" on page 6-2.

- **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 (`pi`), 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 (`pi`), 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 string 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 Conversion Templates" on page 6-2.

- **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 Conversion Templates" on page 6-2.

- **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 stylesheet specifies, which for stylesheets 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 (`pi`), 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 (`pi`), 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 string 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`

Text

Format and insert text into report

Description

This component formats and inserts text into the report. It must have the `Paragraph` component as its parent.

Properties

- **Text to include in report**: Specifies text to include in the report.
- **Style Name**: Specifies the style to use with the 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 more information about template styles, see "Report Conversion Templates" on page 6-2.

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. Suppose that you then apply a text style named `BoldStyle` to text in the paragraph. The text formatted with `BoldStyle` toggles the paragraph style bold off.

One way to make sure that the text is bold is to 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.

%<VariableName> Notation

You can enter `%<VariableName>` in this field (and in any field where the text appears blue) to include the value of a variable from the base MATLAB workspace. You cannot enter more than one variable in `%<>`. If you enter an invalid variable name, the report includes the text `%<VariableName>` instead of the value of the variable.

Example

1. Enter the following text:

```
I have a %<ObjName> and it has %<NumLeaves> leaves.
The word '%<ObjName>' has %<size(ObjName)> letters.
```

2. Set `ObjName` = `"plant"` and `NumLeaves` = 3.
3. Generate the report. It looks as follows:

```
I have a plant and it has 3 leaves.
The word 'plant' has 5 letters.
```

Style

Note: If you use the **Style Name** field to specify a style for this text component, 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.
- **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.
- **Retain spaces and carriage returns**: Formats the text in the 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.
  - 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.

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 month 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. You can use it in a report setup file as a child of a `Chapter/Subsection` component or by itself.

To use the Title Page component, you need to have a Chapter component in your report.

For PDF and HTML reports, you can use the Stylesheet 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 8-16.

Properties

The text fields on this property pane support the `%<VariableName>` notation. For more details, see "%<VariableName> Notation" on page 10-102 on the `Text` component reference page.

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 Conversion Templates" on page 6-2.

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 Conversion Templates" on page 6-2.

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. For more details about this notation, see "%<VariableName> Notation" on page 10-102 on the `Text` component reference page.

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)
`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 a string to evaluate. This string 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 interactively design and generate a report.

Use the Report Explorer to:

- Create and modify report setup files.
- Apply templates or stylesheets 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 2-3
- “Add Report Content Using Components” on page 2-5
- “Generate a Report” on page 2-39

More About

- “Report Setup”
- “Working with Components”
- “Format Reports”
- “Generate Reports”

- “Manage Report Conversion Templates”
- “Customize Report Conversion Templates”

See Also

Functions

`report` | `rptconvert` | `rptlist` | `setedit`

Introduced before R2006a

Functions – Alphabetical List

compwiz

Create custom MATLAB Report Generator components

Syntax

```
compwiz
compwiz ('–browse')
compwiz ('–v1browse')
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 7-2.

- `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 ('–v1browse')` displays a list of legacy (`v1.x`) components from which to derive a new component.
- `compwiz (rptgen.cfr_list)` initializes the Component Editor with the settings of the referenced components.

More About

- “Create Custom Components” on page 7-2

See Also

`setedit` | `report` | `rptconvert` | `rptlist`

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);
```

- “Create Object Containers” on page 13-30
- “Add Content as a Group” on page 13-14

See Also

mlreportgen.dom.Container | mlreportgen.dom.Group

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);
```

- “Create Object Containers” on page 13-30
- “Add Content as a Group” on page 13-14

See Also

mlreportgen.dom.Container | mlreportgen.dom.Group

Introduced in R2015a

mlreportgen.dom.CustomElement.append

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');
```

- “Add Content to a Report” on page 13-11

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.CustomAttribute | mlreportgen.dom.CustomElement

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

string

HTML text, specified as a string.

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);
```

- “Append HTML Content to DOM Reports” on page 13-105

See Also

mlreportgen.dom.HTML | mlreportgen.dom.HTMLFile

More About

- “Appending HTML to DOM Reports” on page 13-103

Introduced in R2015a

addHTMLFile

Class: mlreportgen.dom.Document
Package: mlreportgen.dom

Append HTML file contents to document

Syntax

```
documentObjOut = addHTMLFile(documentObj,htmlFilePath)
```

Description

`documentObjOut = 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

string

HTML file path, specified as a string.

Output Arguments

`documentObjOut` — Document object with HTML file content appended

`mlreportgen.dom.Document` object

Document object with HTML file content appended, returned as an `mlreportgen.dom.Document` 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);
```

- “Append HTML File Contents to DOM Reports” on page 13-107

See Also

mlreportgen.dom.Document.addHTMLFile | mlreportgen.dom.HTML | mlreportgen.dom.HTMLFile

More About

- “Appending HTML to DOM Reports” on page 13-103

Introduced in R2015a

mlreportgen.dom.Document.append

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( ___ ,styleName)

domObjOut = append(docObj,domObj)
```

Description

`domObjOut = append(docObj,textContent)` appends text or numbers to a document and returns a text element object.

`domObjOut = append(docObj,listContent)` appends a one-dimensional cell array as an unordered list and returns an unordered list object.

`domObjOut = append(docObj,tableContent)` appends a two-dimensional cell array as a table and returns a table 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 a document element object.

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 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');
```

- “Add Content to a Report” on page 13-11

Input Arguments

`docObj` — Document to append content to

`mlreportgen.dom.Document` object

Document to append content to, specified as an `mlreportgen.dom.Document` object.

`textContent` — Text to append to document

string

Text to append to document.

Example: `myTextObj = append(myDocument,'This is an introduction')`

`listContent` — Items to append to the document as unordered list

horizontal one-dimensional matrix | cell array

Specify items to be displayed in an unordered list. Each matrix or cell-array element becomes a list item.

For a matrix, use each element can be a numeric or Boolean value.

For a cell array, each element can be:

- A string
- A number
- A Boolean value
- One of these DOM objects:
  - `mlreportgen.dom.Text`
  - `mlreportgen.dom.Paragraph`
  - `mlreportgen.dom.ExternalLink`
  - `mlreportgen.dom.HTML`
  - `mlreportgen.dom.InternalLink`
  - `mlreportgen.dom.Table`
  - `mlreportgen.dom.Image`
  - `mlreportgen.dom.CustomElement`
- A horizontal one-dimensional array (for a sublist)

`tableContent` — Items to append to document as table

matrix | cell array

Specify content for a table using either:

- A vertical one-dimensional matrix or cell array
- A two-dimensional matrix or cell array

For a matrix, each element can be a numeric or Boolean value.

For a cell array, each element can be:

- A string
- A number
- A Boolean value
- One of the following DOM objects:
  - `mlreportgen.dom.Text`
  - `mlreportgen.dom.Paragraph`
  - `mlreportgen.dom.OrderedList`
  - `mlreportgen.dom.UnorderedList`
  - `mlreportgen.dom.Table`
  - `mlreportgen.dom.Image`
  - `mlreportgen.dom.CustomElement`
- A horizontal one-dimensional array (for a sublist)

Example: `{'row 1 - col 1' 'row 1 - col 2';'row 2 - col 1' 'row 2 - col 2'}`

`styleName` — Name of style to apply to appended content

string

The style to use with the appended content. The style defines the appearance of the document element in the output document.

Use a style that is defined in the stylesheet of the template of the document you append content to.

`domObj` — DOM document element object to append

DOM object

You can append the following DOM objects:

- `mlreportgen.dom.CustomElement`
- `mlreportgen.dom.DocumentPart`
- `mlreportgen.dom.DOCXPageLayout`
- `mlreportgen.dom.PDFPageLayout`
- `mlreportgen.dom.DOCXPageHeader`
- `mlreportgen.dom.PDFPageHeader`

- `mlreportgen.dom.DOCXPageFooter`
- `mlreportgen.dom.PDFPageFooter`
- `mlreportgen.dom.FormalTable`
- `mlreportgen.dom.Group`
- `mlreportgen.dom.ExternalLink`
- `mlreportgen.dom.HTML`
- `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.UnorderedList`

Output Arguments

`domObjOut` — Appended document element

document element object

Document element created and appended. One of the following DOM objects:

- `mlreportgen.dom.CustomElement`
- `mlreportgen.dom.DocPart`
- `mlreportgen.dom.DOCXPageLayout`
- `mlreportgen.dom.DOCXPageHeader`
- `mlreportgen.dom.DOCXPageFooter`
- `mlreportgen.dom.PDFPageLayout`
- `mlreportgen.dom.PDFPageHeader`
- `mlreportgen.dom.PDFPageFooter`
- `mlreportgen.dom.FormalTable`

- `mlreportgen.dom.Group`
- `mlreportgen.dom.ExternalLink`
- `mlreportgen.dom.HTML`
- `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.UnorderedList`

See Also

mlreportgen.dom.Document

Introduced in R2014b

mlreportgen.dom.Document.close

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');
```

- “Add Content to a Report” on page 13-11

Input Arguments

`docObj` — Document to close

`mlreportgen.dom.Document` object

Document to close, specified as an `mlreportgen.dom.Document` object.

See Also

mlreportgen.dom.Document | mlreportgen.dom.Document.open

Introduced in R2014b

mlreportgen.dom.Document.createAutoNumberStream

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,myReport.Type);
```

- "Automatically Number Document Content" on page 13-98

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

string

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

string

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.

More About

Tips

When you append an `mlreportgen.dom.AutoNumber` object, specify a numbering stream.

See Also

mlreportgen.dom.Document | mlreportgen.dom.Document.createAutoNumberStream | mlreportgen.dom.Document.getAutoNumberStream

Introduced in R2014b

mlreportgen.dom.Document.createTemplate

Package: mlreportgen.dom

Create DOM template file

Syntax

```
createTemplate(path)
createTemplate(path,type)
```

Description

`createTemplate(path)` creates a copy of the default DOM template file in the specified location. The file extension indicates the type of template file to create.

`createTemplate(path,type)` creates a template file of the specified type in the specified location. If the path does not have an extension, the method appends the extension `.htmtx` for HTML, `.dotx` for Word, `pdftx` for PDF, or `.html` for single-file HTML

Examples

Create a Word Template

Create a Microsoft Word template file in the current folder.

```
import mlreportgen.dom.*
Document.createTemplate('MyWordTemplate','docx');
```

- “Create a Microsoft Word Template” on page 13-124
- “Create an HTML or PDF Template” on page 13-135

Input Arguments

`path` — Path for new template file

string

Path for the template file you want to create, specified as a string. If you use the `path` argument without the `type` argument, include the template extension: `.htmtx` for HTML, `.docx` for Word, `.pdftx`, or `.html` for single-file HTML.

If you use both the `path` and `type` arguments, and you do not specify an extension for `path`, the `createTemplate` method adds the appropriate extension to the new template.

`type` — Type of template

`'html'` (default) | `'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.

The template type must match the type you specified using the `path` argument. If the `path` argument did not include an extension, the `type` argument determines the extension for this method to use.

More About

Tips

Invoke `createTemplate` on the `mlreportgen.dom.Document` class, not on an instance of that class. In other words, use `Document.createTemplate`, not `myDocument.createTemplate`.

See Also

mlreportgen.dom.Document | mlreportgen.dom.Template | `unzipTemplate` | `zipTemplate`

Introduced in R2014b

mlreportgen.dom.Document.fill

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.

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.dotx`.

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);
```

- “Add Content to a Report” on page 13-11

Input Arguments

`form` — Form to fill

string

Form whose holes to fill, specified as a string.

More About

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 | mlreportgen.dom.Document.moveToNextHole

Introduced in R2014b

mlreportgen.dom.Document.getAutoNumberStream

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'
```

- “Automatically Number Document Content” on page 13-98

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

string

Name of the numbering stream to return, specified as a string.

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

mlreportgen.dom.AutoNumber | mlreportgen.dom.AutoNumberStream | mlreportgen.dom.Document.createAutoNumberStream

Introduced in R2014b

mlreportgen.dom.Document.getCoreProperties

Package: mlreportgen.dom

Get document or template core properties

Syntax

```
corePropertiesOut = getCoreProperties(path)
```

Description

`corePropertiesOut = 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

string

Path to the document or template, specified as a string.

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.

More About

- “Report Packages” on page 13-17

See Also

mlreportgen.dom.Document.getOPCMainPart | mlreportgen.dom.Document.setCoreProperties | mlreportgen.dom.OPCPart

Introduced in R2014b

mlreportgen.dom.Document.getImageDirectory

Package: mlreportgen.dom

Get image folder of document

Syntax

```
imageDirectory = getImageDirectory(path,type)
```

Description

`imageDirectory = 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.htmx` 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

string

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

`imageDirectory` — Image folder of document

string

The path of the image folder for the package.

More About

- “Report Packages” on page 13-17

See Also

mlreportgen.dom.Document.getImagePrefix | mlreportgen.dom.Document.getOPCMainPart | mlreportgen.dom.Document.setCoreProperties | mlreportgen.dom.OPCPart

Introduced in R2014b

mlreportgen.dom.Document.getImagePrefix

Package: mlreportgen.dom

Get generated image name prefix

Syntax

```
imagePrefix = getImagePrefix(path,type)
```

Description

`imagePrefix = 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

string

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

string

The generated image name prefix.

More About

- “Report Packages” on page 13-17

See Also

mlreportgen.dom.Document.getImageDirectory | mlreportgen.dom.Document.getOPCMainPart | mlreportgen.dom.Document.setCoreProperties | mlreportgen.dom.OPCPart

Introduced in R2014b

mlreportgen.dom.Document.getMainPartPath

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.hml'
```

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
string

Path of the main part of document output package.

More About

- "Report Packages" on page 13-17

See Also

mlreportgen.dom.Document.getOPCMainPart | mlreportgen.dom.Document.setCoreProperties | mlreportgen.dom.OPCPart

Introduced in R2014b

mlreportgen.dom.Document.getOPCMainPart

Package: mlreportgen.dom

Return main part of document, document part, or template

Syntax

```
partOut = getOPCMainPart(path)
partOut = getOPCMainPart(path,docType)
```

Description

`partOut = 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 = 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

string

If you use the `path` argument without the `docType` argument, include the `.docx` or `.htmx` extension.

If you use both the `path` and `docType` arguments, `getOPCMainPart` appends an extension of the appropriate type (`.docx` or `.htmx`).

`docType` — Type of document, document part, or template

`'docx' | 'html'`

Type of document, document part, or template, specified as a string.

Output Arguments

`partOut` — Path of main part of a package

string

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 (/).

More About

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.

- “Report Packages” on page 13-17

See Also

mlreportgen.dom.Document.getMainPartPath | mlreportgen.dom.Document.setCoreProperties | mlreportgen.dom.OPCPart

Introduced in R2014b

mlreportgen.dom.Document.moveToNextHole

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);
```

- “Add Content to a Report” on page 13-11

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`

More About

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 | mlreportgen.dom.Document.fill

Introduced in R2014b

mlreportgen.dom.Document.open

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);
```

• "Add Content to a Report" on page 13-11

Input Arguments

`docObj` — Document to open

`mlreportgen.dom.Document` object

Document to open, specified as an `mlreportgen.dom.Document` object.

More About

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

mlreportgen.dom.Document | mlreportgen.dom.Document.close

Introduced in R2014b

mlreportgen.dom.Document.package

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.

More About

- "Report Packages" on page 13-17

See Also

mlreportgen.dom.Document.getCoreProperties | mlreportgen.dom.Document.getOPCMainPart | mlreportgen.dom.OPCPart | `unzipTemplate` | `zipTemplate`

Introduced in R2014b

mlreportgen.dom.Document.setCoreProperties

Package: mlreportgen.dom

Set OPC core properties of output document or template

Syntax

```
corePropertiesOut = setCoreProperties(path,corePropertiesObj)
```

Description

`corePropertiesOut = 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

string

Path of document, document part, or template, specified as a string.

`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.

More About

- "Report Packages" on page 13-17

See Also

mlreportgen.dom.CoreProperties | mlreportgen.dom.Document.getCoreProperties | mlreportgen.dom.Document.getOPCMainPart

Introduced in R2014b

mlreportgen.dom.ExternalLink.append

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 string 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.

Examples

• "Create Links" on page 13-83

Input Arguments

`externalLinkObj` — External link object to append custom element to

`mlreportgen.dom.ExtermalLink` object

External link object to append custom element to, specified as an `mlreportgen.dom.ExtermalLink` object.

`text` — Text to append

string

Text to append, specified as a string.

`styleName` — Name of style to apply to appended text string

string

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 stylesheet 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.InternalLink | mlreportgen.dom.LinkTarget

Introduced in R2014b

mlreportgen.dom.FormalTable.appendFooterRow

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');
```

- “Create and Format Tables” on page 13-69

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.FormalTable | mlreportgen.dom.Table

Introduced in R2014b

mlreportgen.dom.FormalTable.appendHeaderRow

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');
```

- “Create and Format Tables” on page 13-69

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.FormalTable | mlreportgen.dom.Table

Introduced in R2014b

mlreportgen.dom.Group.append

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');
```

- “Add Content to a Report” on page 13-11

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.Document | mlreportgen.dom.DocumentPart | mlreportgen.dom.DOCXPageLayout | mlreportgen.dom.Group | mlreportgen.dom.PDFPageLayout

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

string

HTML text, specified as a string

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);
```

- “Append HTML Content to DOM Reports” on page 13-105

See Also

mlreportgen.dom.Document.addHTML | mlreportgen.dom.HTML | mlreportgen.dom.HTMLFile

More About

- “Appending HTML to DOM Reports” on page 13-103

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);
```

- “Append HTML File Contents to DOM Reports” on page 13-107

See Also

mlreportgen.dom.HTML | mlreportgen.dom.HTMLFile

More About

- “Appending HTML to DOM Reports” on page 13-103

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 string

string

HTML text, specified as a string.

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 HTML Object

This example assumes that there is an HTML file called `myHTMLfile.html` in the MATLAB current folder.

Create an `HTMLFile` object from HTML text to use for a Microsoft Word report.

```
import mlreportgen.dom.*;
rpt = Document('HTMLToWordReport','docx');
htmlFileObj = HTML('<p><b>Hello</b> <i style="color:green">World</i></p>');
```

Append content to the `HTMLFile` object and append the `HTMLFile` object to the document.

```
append(htmlFileObj,'<p>This is <u>me</u> speaking</p>');
append(rpt,htmlFileObj);
```

Generate the report.

```
close(rpt);
rptview(rpt.OutputPath);
```

- “Append HTML File Contents to DOM Reports” on page 13-107

See Also

mlreportgen.dom.HTML | mlreportgen.dom.HTMLFile

More About

- “Appending HTML to DOM Reports” on page 13-103

Introduced in R2015a

mlreportgen.dom.LinkTarget.append

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

```
import mlreportgen.dom.*
d = Document('mydoc');

target = LinkTarget('home')
append(target,' - top of page')
```

```
append(d,target);
append(d,InternalLink('home','Go to Top');

close(d);
rptview('mydoc', 'html');
```

Append an Automatically Generated Number to a Link Target

```
import mlreportgen.dom.*
d = Document('mydoc','docx');

number = AutoNumber('paragraph');

target = LinkTarget('para');
append(target,'number');
InternalLink('para','Link to paragraph');
q = Paragraph('This paragraph is not linked to.');
append(d,q);
PageBreakBefore(true);
p = Paragraph('This is the paragraph that is linked to.');
append(p,
p.Style = {CounterInc('paragraph'),PageBreakBefore(true)};

append(d,p);

close(d);
rptview('mydoc','docx');
```

- “Add Content to a Report” on page 13-11

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

string

Text to append, specified as a string.

`styleName` — Name of style
string

Name of style, specified as a string.

`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 `mlreportgen.dom.AutoNumber` object.

Output Arguments

`textObj` — Text object
`mlreportgen.dom.Text` object

Text object, represented by an `mlreportgen.dom.Text` object.

`autoNumberObj` — Automatically generated number for link target
`mlreportgen.dom.AutoNumber` object

Automatically generated number for link target, specified as a string.

See Also

mlreportgen.dom.AutoNumber | mlreportgen.dom.Text

Introduced in R2014b

mlreportgen.dom.MessageDispatcher.dispatch

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.

• "Display Report Generation Messages" on page 13-119

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

Introduced in R2014b

mlreportgen.dom.MessageDispatcher.getTheDispatcher

Package: mlreportgen.dom

Return DOM message dispatcher

Syntax

```
getTheDispatcher
```

Description

`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.

- “Display Report Generation Messages” on page 13-119

See Also

mlreportgen.dom.MessageDispatcher.dispatch | mlreportgen.dom.MessageEventData | mlreportgen.dom.MessageFilter

Introduced in R2014b

mlreportgen.dom.OrderedList.append

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,ul);

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');
```

- “Create and Format Lists” on page 13-63

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 string
- 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.ListItem | mlreportgen.dom.OrderedList | mlreportgen.dom.UnorderedList

Introduced in R2014b

mlreportgen.dom.Paragraph.append

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 string 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('mydoc','html');
```

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...
   ('http://www.mathworks.com/','MathWorks site');
para = Paragraph('Go to the ');
append(para, mathWorksLink);
append(docLink,para);

close(docLink);
rptview('mydocLink','html');
```

- “Add Content to a Report” on page 13-11

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 string to append to paragraph

string

Text string to append to the paragraph, specified as a string.

`styleName` — Name of a style to apply to text

string

Name of the style to define the appearance of the text. Use a style that is in the stylesheet 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 | mlreportgen.dom.Paragraph.clone | mlreportgen.dom.Text

Introduced in R2014b

mlreportgen.dom.Paragraph.clone

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.*;
myReport = 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: []
           Children: [1x1 mlreportgen.dom.Text]
             Parent: []
                Tag: 'dom.Paragraph:15'
                 Id: '15'
```

- “Add Content to a Report” on page 13-11

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.

More About

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

mlreportgen.dom.Document | mlreportgen.dom.Paragraph | mlreportgen.dom.Paragraph.append

Introduced in R2014b

mlreportgen.dom.ProgressMessage.formatAsHTML

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);
```

- “Display Report Generation Messages” on page 13-119

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

mlreportgen.dom.MessageFilter | mlreportgen.dom.ProgressMessage | mlreportgen.dom.ProgressMessage.formatAsText

Introduced in R2014b

mlreportgen.dom.ProgressMessage.formatAsText

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);
```

- “Display Report Generation Messages” on page 13-119

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

mlreportgen.dom.MessageFilter | mlreportgen.dom.ProgressMessage | mlreportgen.dom.ProgressMessage.formatAsHTML

Introduced in R2014b

mlreportgen.dom.ProgressMessage.passesFilter

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.

- "Display Report Generation Messages" on page 13-119

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

Introduced in R2014b

mlreportgen.dom.Table.entry

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.*;
myReport = Document('myDoc','html');
t = Table(magic(5));
t.entry(3,4);
t.entry(3,4).Children(1).Color = 'red';
append(myReport,t);
close(myReport);
rptview('myDoc','html');
```

- “Create and Format Tables” on page 13-69

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.Table.row | mlreportgen.dom.TableEntry

Introduced in R2014b

mlreportgen.dom.Table.row

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('myDoc','html');
```

- “Create and Format Tables” on page 13-69

Input Arguments

`tableObj` — Table containing entry

`mlreportgen.dom.Table` object | `mlreportgen.dom.FormalTable` object

Table containing the entry, specified as an `mlreportgen.dom.Table` or `mlreportgen.dom.FormalTable` 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.TableEntry | mlreportgen.dom.TableRow

Introduced in R2014b

mlreportgen.dom.TableRow.append

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('myDoc','html');
```

- “Create and Format Tables” on page 13-69

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.TableEntry | mlreportgen.dom.TableRow

Introduced in R2014b

mlreportgen.ppt.ContentPlaceholder.add

Package: mlreportgen.ppt

Add paragraph or paragraphs to content placeholder

Syntax

```
paraObj = add(contentPlaceholder,content)
add(contentPlaceholder,contents)
```

Description

`paraObj = add(contentPlaceholder,content)` adds a paragraph to a content placeholder.

`add(contentPlaceholder,contents)` adds multiple paragraphs to a content placeholder.

Examples

Add Paragraphs to a Content Placeholder

Create a presentation.

```
import mlreportgen.ppt.*
slidesFile = 'myContentPlaceholder.pptx';
slides = Presentation(slidesFile);
add(slides,'Title and Content');
```

Use the `Presentation.find` method to find the `Content` placeholders.

```
contents = find(slides,'Content');
```

Create an `mlreportgen.ppt.Paragraph` object. Add the paragraph to the first `Content` placeholder.

```
para = add(contents(1),'First item in the list' );
```

```
para.Italic = true;
para.FontColor = 'green';
```

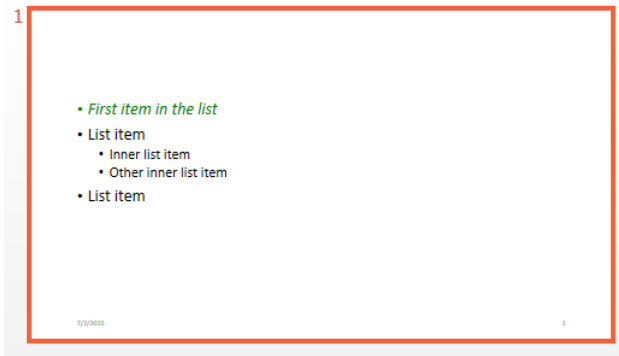
Add multiple paragraphs to the first `Content` placeholder.

```
add(contents(1),{ ...
    'List item', ...
        {'Inner list item','Other inner list item' }...
    'List item', ...
    });
```

Generate the presentation and then open `myBoldPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
close(slides);

if ispc
    winopen(slidesFile);
end
```



- “Access PowerPoint Template Elements” on page 14-37
- “Add and Replace Presentation Content” on page 14-74

Input Arguments

`contentPlaceholder` — Content placeholder to add content to

`mlreportgen.ppt.ContentPlaceholder` object

Content placeholder to add content to, specified as an `mlreportgen.ppt.ContentPlaceholder` object.

`content` — Content to add

string | `mlreportgen.ppt.Paragraph` object

Content to add, specified as a string or an `mlreportgen.ppt.Paragraph` object. The slide layout specifies whether the text displays as a paragraph, bulleted list item, or numbered list item.

`contents` — Multiple paragraphs to add

cell array of strings or `mlreportgen.ppt.Paragraph` objects or a combination of both | cell array of `mlreportgen.ppt.Paragraph` objects | cell array of a combination of strings and `mlreportgen.ppt.Paragraph` objects

Content to add, specified as a cell array of strings, `mlreportgen.ppt.Paragraph` objects, or a combination of both. Inner cell arrays specify inner list items. The slide layout specifies whether the text displays as paragraphs, bulleted list items, or numbered list items.

Output Arguments

`paraObj` — Paragraph added to content placeholder

`mlreportgen.ppt.Paragraph` object

Paragraph added to content placeholder, returned as an `mlreportgen.ppt.Paragraph` object.

See Also

mlreportgen.ppt.ContentPlaceholder | mlreportgen.ppt.ContentPlaceholder.replace | mlreportgen.ppt.Paragraph

Introduced in R2015b

mlreportgen.ppt.ContentPlaceholder.replace

Package: mlreportgen.ppt

Replace content placeholder or its content

Syntax

```
contentObj = replace(ContentPlaceholder,content)
replace(ContentPlaceholder,paragraphs)
```

Description

`contentObj = replace(ContentPlaceholder,content)` replaces the content of the `ContentPlaceholder` object or replaces the `ContentPlaceholder` object. If `content` is a paragraph, it replaces the placeholder content. If `content` is a table or a picture, it replaces the placeholder object.

`replace(ContentPlaceholder,paragraphs)` replaces the content of a content placeholder with multiple paragraphs.

Examples

Replace Content Placeholders Using Paragraph and Table Objects

The type of object you use to replace a content placeholder determines whether the object replaces the placeholder's content or the entire placeholder. Using a paragraph replaces the content. Using a table or a picture replaces the entire placeholder. This example shows how to use the `mlreportgen.ppt.ContentPlaceholder.replace` method and what happens to the `ContentPlaceholder` object based on the object you replace it with.

Create a presentation and add two Title and Content slides from the default template. Return the instances of the `ContentPlaceholder` object `'Content'` and assign them to a variable. The `contents` variable is an array.

```
import mlreportgen.ppt.*

slides = Presentation('MyPresentation');

add(slides,'Title and Content');
add(slides,'Title and Content');

contents = find(slides,'Content');
```

Replace the first `'Content'` object in the array using a table. Replace the second `'Content'` object using a paragraph. Return the updated instances of the `'Content'` objects and update the variable with the new objects.

```
replace(contents(1),Table(magic(5)));
replace(contents(2),Paragraph('Hello'));

contents = find(slides,'Content');
```

Display the class of each of the `'Content'` objects in the `contents` array. The placeholder you replaced using a table is a `Table` object. The placeholder you replaced using a paragraph is a `ContentPlaceholder` object that contains the new paragraph.

```
disp(class(contents(1)))
```

```
mlreportgen.ppt.Table
```

```
disp(class(contents(2)))
```

```
mlreportgen.ppt.ContentPlaceholder
```

Because the second object is still a placeholder object, you can replace the contents again.

```
replace(contents(2),Paragraph('Goodbye'));
```

Because the second object is no longer a `ContentPlaceholder` object, you can replace it only with an object of the same type, in this case a table. Replacing it with a paragraph or picture returns an error.

Replace the Content Placeholder Content with Multiple Paragraphs

Create a presentation.

```
import mlreportgen.ppt.*
```

```
name1 = 'before';
slides = Presentation(name1);
add(slides,'Comparison');
```

The default PPT API PowerPoint template `Comparison` slide layout has a `Left Content` and a `Right Content` placeholder. Replace the content in those content placeholders. Then generate the presentation.

```
replace(slides,'Left Content','dummy content');
replace(slides,'Right Content','dummy content');
close(slides);
```

Create a second presentation, using the first presentation as the template.

```
name2 = 'after';
slides = Presentation(name2,name1);
```

Use the `find` method with the `Presentation` object to return content objects named `Left Content` and `Right Content`.

```
left = find(slides,'Left Content');
right = find(slides,'Right Content');
```

Replace the left and right content.

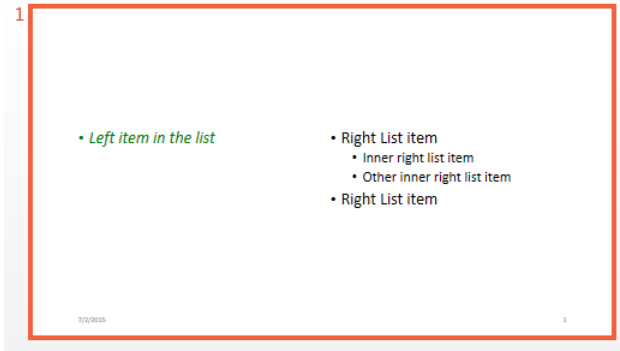
```
para = replace(left(1),'Left item in the list');
para.Italic = true;
para.FontColor = 'green';

replace(right(1), { ...
    'Right List item', ...
        {'Inner right list item','Other inner right list item'}...
    'Right List item', ...
    });
```

Generate the presentation and then open `myBoldPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
close(slides);

if ispc
    winopen(slides.OutputPath);
end
```



Input Arguments

`ContentPlaceholder` — Content placeholder to replace or whose content to replace

`mlreportgen.ppt.ContentPlaceholder` object

Content placeholder to replace content or whose content to replace, specified as an `mlreportgen.ppt.ContentPlaceholder` object.

`content` — Content to use as replacement

string | `mlreportgen.ppt.Paragraph` object | `mlreportgen.ppt.Table` object | `mlreportgen.ppt.Picture` object

Content to use as replacement, specified as a string or one of these objects:

- `mlreportgen.ppt.Paragraph`
- `mlreportgen.ppt.Table`
- `mlreportgen.ppt.Picture`

`paragraphs` — Paragraphs to replace placeholder content with

cell array of strings | cell array of `mlreportgen.ppt.Paragraph` objects | cell array of a combination of strings and `mlreportgen.ppt.Paragraph` objects

Paragraphs to replace placeholder content with, specified as a cell array of strings, `mlreportgen.ppt.Paragraph` objects, or a combination of both. Inner cell arrays specify inner list items. The slide layout determines whether the text displays as paragraphs, bullet list items, or numbered list items.

Example: `{'My first paragraph','My second paragraph'}`

Output Arguments

`contentObj` — Content object

`mlreportgen.ppt.Paragraph` object | `mlreportgen.ppt.Table` object | `mlreportgen.ppt.Picture` object

Content object, returned as an `mlreportgen.ppt.Paragraph`, `mlreportgen.ppt.Table`, or `mlreportgen.ppt.Picture` object. The output object corresponds to the string or content object that you specify with the `content` input argument.

See Also

mlreportgen.ppt.ContentPlaceholder | mlreportgen.ppt.ContentPlaceholder.add | mlreportgen.ppt.Paragraph | mlreportgen.ppt.Picture | mlreportgen.ppt.Table

Introduced in R2015b

mlreportgen.ppt.MessageDispatcher.dispatch

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.

• "Display Presentation Generation Messages" on page 14-15

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

mlreportgen.ppt.MessageDispatcher.getTheDispatcher | mlreportgen.ppt.MessageEventData | mlreportgen.ppt.MessageFilter

Introduced in R2015b

mlreportgen.ppt.MessageDispatcher.getTheDispatcher

Package: mlreportgen.ppt

Return PPT message dispatcher

Syntax

```
getTheDispatcher
```

Description

`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.

- "Display Presentation Generation Messages" on page 14-15

See Also

mlreportgen.ppt.MessageDispatcher.dispatch | mlreportgen.ppt.MessageEventData | mlreportgen.ppt.MessageFilter

Introduced in R2015b

mlreportgen.ppt.Paragraph.append

Package: mlreportgen.ppt

Append content to paragraph

Syntax

```
contentObj = append(paragraph,content)
```

Description

`contentObj = append(paragraph,content)` appends content to a paragraph.

Examples

Append Formated Text and External Link to Paragraph

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myParagraphPresentation.pptx'
slides = Presentation(slidesFile);
add(slides,'Title Slide');
add(slides,'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(slides,'Title',p1);
```

Create a paragraph for the content of the second slide.

```
p2 = Paragraph('Click the link for the ');
contentObj = append(p2,ExternalLink('http://www.mathworks.com','MathWorks site.'));
```

Replace the content with the `p2` paragraph.

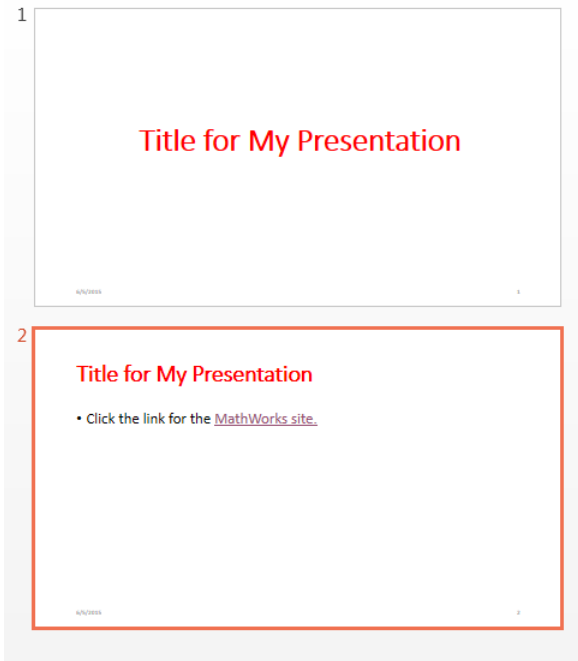
```
replace(slides,'Content',p2);
```

Close the presentation.

```
close(slides);
```

Open `myParagraphPresentation.pptx` file. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```



- “Add and Replace Text” on page 14-75

- “Create and Format Paragraphs” on page 14-84

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

string | `mlreportgen.ppt.Text` object | `mlreportgen.ppt.ExternalLink` object

Content to add to a paragraph, specified as a string, an `mlreportgen.ppt.Text` object, or an `mlreportgen.ppt.ExternalLink` object.

Output Arguments

`contentObj` — Content object

`mlreportgen.ppt.Text` object | `mlreportgen.ppt.ExternalLink` object

Content object, returned as an `mlreportgen.ppt.Text` or `mlreportgen.ppt.ExternalLink` object. The output object corresponds to the input string or `Text` object.

See Also

mlreportgen.ppt.ExternalLink | mlreportgen.ppt.Paragraph | mlreportgen.ppt.Text

Introduced in R2015b

mlreportgen.ppt.Picture.replace

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.*

slides = Presentation('myPictureReplacePresentation');
slide1 = add(slides,'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 the presentation.

```
close(slides);
```

- “Access PowerPoint Template Elements” on page 14-37
- “Add or Replace a Picture” on page 14-79

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

Introduced in R2015b

mlreportgen.ppt.PicturePlaceholder.replace

Package: mlreportgen.ppt

Replace picture in picture placeholder

Syntax

```
pictureObj = replace(picturePlaceholder,picture)
```

Description

`pictureObj = replace(picturePlaceholder,picture)` replaces the picture in a picture placeholder. PowerPoint modifies the picture dimensions to fill in the `picturePlaceholder` dimensions. If the picture dimensions are bigger, PowerPoint stretches the image proportionally. If the picture dimensions are smaller, the picture is centered.

Examples

Replace Picture in Picture Placeholder

Create a presentation.

```
import mlreportgen.ppt.*
slidesFile = 'myPlaceholderPresentation.pptx';
slides = Presentation(slidesFile);
slide1 = add(slides,'Title and Picture');
```

Create an `mlreportgen.ppt.Picture` object.

```
plane = Picture(which('b747.jpg'));
```

Find an object whose `Name` property is `Picture`.

```
pictures = find(slide1,'Picture');
```

Replace the picture in the picture placeholder.

```
replace(pictures(1),plane);
```

Generate the presentation and then open `myPlaceholderPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
close(slides);

if ispc
    winopen(slidesFile);
end
```

- "Access PowerPoint Template Elements" on page 14-37
- "Add or Replace a Picture" on page 14-79

Input Arguments

`picturePlaceholder` — Picture placeholder whose picture to replace

`mlreportgen.ppt.PicturePlaceholder` object

Picture placeholder whose picture to replace, specified as an `mlreportgen.ppt.PicturePlaceholder` object.

`picture` — Picture to use as replacement

`mlreportgen.ppt.Picture` object

Picture to use as replacement, specified as an `mlreportgen.ppt.Picture` object.

Output Arguments

`pictureObj` — Picture

`mlreportgen.ppt.Picture` object

Picture, represented by an `mlreportgen.ppt.Picture` object.

See Also

mlreportgen.ppt.Picture | mlreportgen.ppt.PicturePlaceholder

Introduced in R2015b

mlreportgen.ppt.Presentation.add

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 under the first slide master in the presentation.

`slideObj = add(presentation,slideLayout,slideMaster)` uses the slide layout under the specified slide master. Use the `slideMaster` argument when the presentation contains multiple slide masters. If you do not provide the `slideMaster` argument, a new slide is created from the first layout that matches with `slideLayout` name.

`slideObj = add(presentation,slideLayout,otherSlide)` adds the slide immediately before the slide specified in the `otherSlide` argument, using the specified slide layout under 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 under the specified slide master.

`slideObj = add(presentation,slideLayout,index)` adds the slide at the index position specified by `index`, using the specified slide layout under the specified slide master.

`slideObj = add(presentation,slideLayout,slideMaster,index)` adds the slide immediately before the slide specified by `otherSlide`, using the specified slide layout under the specified slide master.

Examples

Add Slides

Create a presentation.

```
import mlreportgen.ppt.*
slidesFile = 'myAddSlidesPresentation.pptx';
slides = Presentation(slidesFile);
```

Add a first slide, specifying the slide layout, but not the slide master or location.

```
contentSlide = add(slides,'Title and Content');
replace(contentSlide,'Title','This is the Title of the Slide Content');
```

Add another slide using the `Office Theme` slide master. Have it appear before `contentSlide`.

```
titleSlide = add(slides,'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.

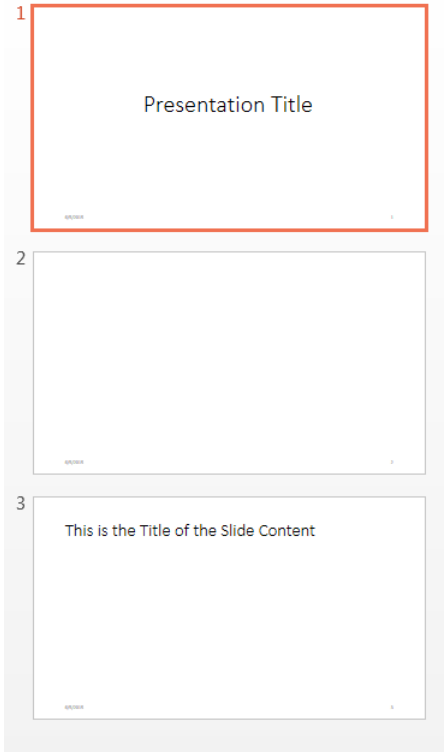
```
blankSlide = add(slides,'Blank','Office Theme',2);
```

Close presentation.

```
close(slides);
```

Open `myAddSlidesPresentation.pptx` file. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```



- “Add and Replace Presentation Content” on page 14-74

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

string

Layout of slide to add, specified as a string. The layout must be in the presentation template.

To see the available layouts, you can:

- Use the `mlreportgen.ppt.Presentation.getLayoutNames` method.
- In the PowerPoint template, select **Home** > **Layout**.

`slideMaster` — Slide master for the slide layout
string

Slide master for the specified slide layout. The slide master must be in the presentation template.

To see the available layouts, use one of these approaches:

- Use the `mlreportgen.ppt.Presentation.getSlideMasterNames` method.
- In the PowerPoint template, select **View** > **Slide Master**. The slide masters are the numbered slides. To get the name of a slide master, hover over it. 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 representing position of slide in presentation
double

Index representing position of slide in presentation, specified as a double.

Output Arguments

`slideObj` — Slide
`mlreportgen.ppt.Slide` object

Slide, returned as an `mlreportgen.ppt.Slide` object.

See Also

mlreportgen.ppt.Presentation | mlreportgen.ppt.Presentation.getLayoutNames | mlreportgen.ppt.Presentation.replace | mlreportgen.ppt.Slide

Introduced in R2015b

mlreportgen.ppt.Presentation.close

Package: mlreportgen.ppt

Close presentation

Syntax

```
close(presentation)
```

Description

`close(presentation)` closes a presentation and generates a PowerPoint presentation.

Examples

Create and Generate a Presentation

Create a presentation.

```
import mlreportgen.ppt.*
slidesFile = 'myParagraphPresentation.pptx';
slides = Presentation(slidesFile);

add(slides,'Title Slide');
```

Create a `Paragraph` object to use for the title of slides. 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(slides,'Title');
replace(contents(1),p);
```

Close the presentation.

```
close(slides);
```

Open `myParagraphPresentation.pptx` file. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```

- “Generate a Presentation” on page 14-14

Input Arguments

`presentation` — Presentation to close and generate

`mlreportgen.ppt.Presentation` object

Presentation to close, specified as an `mlreportgen.ppt.Presentation` object.

See Also

mlreportgen.ppt.Presentation | mlreportgen.pptPresentation.open

Introduced in R2015b

mlreportgen.ppt.Presentation.find

Package: mlreportgen.ppt

Search in presentation

Syntax

```
searchResults = find(presentation,objectName)
```

Description

`searchResults = find(presentation,objectName)` searches in the presentation for the content objects whose `Name` property value matches the value you specify using `objectName`.

Examples

Search in Presentation

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myPresFindPresentation.pptx';
slides = Presentation(slidesFile);
add(slides,'Title Slide');
add(slides,'Title and Content');
```

Use the `Presentation.find` method to find presentation objects whose `Name` property is `Title`.

```
contents = find(slides,'Title')
```

```
contents =

  1x2 TextBoxPlaceholder array with properties:

    Bold
```

```
    FontColor
    Italic
    Strike
    Subscript
    Superscript
    Underline
    Name
    X
    Y
    Width
    Height
    Style
    Children
    Parent
    Tag
    Id
```

Create a paragraph.

```
p = Paragraph('My Presentation Title');
```

Replace the content in slide 1 with `Paragraph` object `p`.

```
replace(contents(1),p);
```

Generate the presentation. Open `myPresFindPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
close(slides);

if ispc
    winopen(slidesFile);
end
```

- “Add and Replace Presentation Content” on page 14-74

Input Arguments

`presentation` — Presentation to search

`mlreportgen.ppt.Presentation` object | string

Presentation to search, specified as an `mlreportgen.ppt.Presentation` object or as a string that contains the name of a presentation.

`objectName` — Name property value to search

string

Name property value to search, specified as string. The search looks in the whole presentation, searching for objects with the specified `Name` property value. Search strings are case-sensitive. The search treats the string as a full string.

Output Arguments

`searchResults` — Search results

object array

Search results, returned as an object array.

See Also

mlreportgen.ppt.Presentation | mlreportgen.ppt.Slide.find

Introduced in R2015b

mlreportgen.ppt.Presentation.getLayoutNames

Package: mlreportgen.ppt

Get names of layouts for presentation slide master

Syntax

```
layoutNames = getLayoutNames(presentation,slideMaster)
```

Description

`layoutNames = getLayoutNames(presentation,slideMaster)` gets the names of layouts for a presentation slide master.

Examples

Get Names of Slide Layouts

Create a presentation.

```
import mlreportgen.ppt.*
slidesFile = 'layouts.pptx';
slides = Presentation(slidesFile);
```

Get the names of the slide masters in the presentation.

```
masters = getMasterNames(slides);
```

Get the names of layouts in the first slide master.

```
layouts = getLayoutNames(slides,masters{1});
layouts
```

```
Columns 1 through 5

    'Title Slide' 'Title and Vertica…'  'Vertical Title an…'  'Title and Table'  'Title
```

```
  Columns 6 through 11

    'Title and Content'   'Section Header'   'Two Content'   'Comparison'   'Title Only'

  Columns 12 through 13

    'Content with Capt…'   'Picture with Capt…'
```

Generate the presentation. Open `layouts.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
close('slides')

if ispc
    winopen(slidesFile);
end
```

- "Access PowerPoint Template Elements" on page 14-37
- "Set Up a PowerPoint Template" on page 14-26

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

string

Slide master to get layout names for, specified as a string.

Output Arguments

`layoutNames` — Slide layout names

cell array of strings

Slide layout names in the PowerPoint template, returned as a cell array of strings.

See Also

mlreportgen.ppt.Presentation | mlreportgen.pptPresentation.getMasterName | mlreportgen.pptPresentation.getTableStyleNames

Introduced in R2015b

mlreportgen.ppt.Presentation.getMasterNames

Package: mlreportgen.ppt

Get names of slide masters for presentation

Syntax

```
masters = getMasterNames(presentation)
```

Description

`masters = getMasterNames(presentation)` gets the names of slide masters for a presentation.

Examples

Get Slide Master Names of Default Template

Create a presentation.

```
import mlreportgen.ppt.*
slides = Presentation('myGetMastersPresentation');
```

Get the names of master slides in the presentation. The default PPT API template has just one slide master.

```
masters = getMastersNames(slides);
masters(1)
```

```
ans =

    'Office Theme'
```

- “Access PowerPoint Template Elements” on page 14-37
- “Set Up a PowerPoint Template” on page 14-26

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 strings

Slide masters in presentation, returned as a cell array of strings.

See Also

mlreportgen.ppt.Presentation | mlreportgen.pptPresentation.getLayoutName | mlreportgen.pptPresentation.getTableStyleNames

Introduced in R2015b

mlreportgen.ppt.Presentation.getTableStyleNames

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

Create a presentation.

```
import mlreportgen.ppt.*
slides = Presentation('myPlaceholderPresentation');
```

Get the names of table styles in the presentation. The output actually includes many more table style names.

```
getTableStyleNames(slides)
```

```
ans =

    '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}'
```

Create a table with the specified table format.

```
table1 = Table({'a','b';'c','d'},'Medium Style 2 - Accent 1');
```

- “Access PowerPoint Template Elements” on page 14-37
- “Set Up a PowerPoint Template” on page 14-26

Input Arguments

`presentation` — Presentation to get table style names for

`mlreportgen.ppt.Presentation` object

Presentation to get tables style names for, specified as an `mlreportgen.ppt.Presentation` object.

Output Arguments

`tableStyles` — Table styles in presentation

n-by-2 cell array of strings

Table styles in the presentation, returned as an array of strings. Each table style name has an associated text string name and numeric string identifier. For example:

```
'Medium Style 2 - Accent 1'           '{5C22544A-7EE6-4342-B048-85BDC9FD1C3A}'
```

To use a table style name with the PPT API, you can use either the name string or the numeric identifier string.

See Also

mlreportgen.ppt.Presentation | mlreportgen.pptPresentation.getLayoutNames | mlreportgen.pptPresentation.getMasterNames

Introduced in R2015b

mlreportgen.ppt.Presentation.open

Package: mlreportgen.ppt

Open presentation

Syntax

```
open(presentation)
```

Description

`open(presentation)` opens a presentation and parses the template. To create an custom template based on the PPT API default template, you can create an empty presentation and open and close the empty template without adding any slides.

Examples

Open a Presentation

Create a presentation.

```
import mlreportgen.ppt.*
slides = Presentation('myTemplate');
```

Open the presentation.

```
open(slides);
```

Close the presentation.

```
close(slides);
```

- “Create a Presentation Program” on page 14-2
- “Generate a Presentation” on page 14-14

Input Arguments

`presentation` — Presentation to open

`mlreportgen.ppt.Presentation` object

Presentation to open, specified as an `mlreportgen.ppt.Presentation` object.

See Also

mlreportgen.dom.Presentation | mlreportgen.pptPresentation.close

Introduced in R2015b

mlreportgen.ppt.Presentation.replace

Package: mlreportgen.ppt

Replace paragraphs, tables, or pictures in presentation

Syntax

```
replace(presentation,placeholderName,content)
```

Description

`replace(presentation,placeholderName,content)` replaces existing content of a placeholder with one or more paragraphs, a table, or a picture that you specify. If the type of content you specify in the content input argument is not valid for replacing the placeholder, the `replace` method has no effect.

Examples

Replace Presentation Content

Create a presentation.

```
import mlreportgen.ppt.*
slides = Presentation('myFirstPresentation');
```

Add slides.

```
add(slides,'Title Slide');
add(slides,'Title and Content');
```

Replace all the titles in the presentation.

```
replace(slides,'Title','My Slide Title');
```

Generate the presentation.

```
close(slides);
```

- “Add and Replace Presentation Content” on page 14-74

Input Arguments

`presentation` — Presentation to replace content in

`mlreportgen.ppt.Presentation` object

Presentation to replace content in, specified as an `mlreportgen.ppt.Presentation` object.

`placeholderName` — Name of placeholders to replace

string

Name of placeholders to replace, specified as a string. The placeholder name must be defined in the template.

`content` — Content to use as replacement

string | `mlreportgen.ppt.Paragaph` object | cell array of strings or `Paragraph` objects, or a combination of both | `mlreportgen.ppt.Table` object | `mlreportgen.ppt.Picture` object

Content to use as replacement, specified as one of these:

- string
- `mlreportgen.ppt.Paragaph` object
- cell array of strings or `Paragraph` objects, or a combination of both
- `mlreportgen.ppt.Table` object
- `mlreportgen.ppt.Picture` object

See Also

mlreportgen.ppt.Presentation | mlreportgen.ppt.Presentation.add | mlreportgen.ppt.Presentation.getLayoutNames | mlreportgen.ppt.Slide

Introduced in R2015b

mlreportgen.ppt.ProgressMessage.formatAsHTML

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);
```

- “Display Presentation Generation Messages” on page 14-15

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

mlreportgen.ppt.MessageEventData | mlreportgen.ppt.MessageFilter | mlreportgen.ppt.ProgressMessage | mlreportgen.ppt.ProgressMessage.formatAsText

Introduced in R2015b

mlreportgen.ppt.ProgressMessage.formatAsText

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);
```

- “Display Presentation Generation Messages” on page 14-15

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

mlreportgen.ppt.MessageEventData | mlreportgen.ppt.MessageFilter | mlreportgen.ppt.ProgressMessage | mlreportgen.ppt.ProgressMessage.formatAsHTML

Introduced in R2015b

mlreportgen.ppt.ProgressMessage.passesFilter

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.

- "Display Presentation Generation Messages" on page 14-15

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.MessageEventData | mlreportgen.ppt.MessageFilter | mlreportgen.ppt.ProgressMessage

Introduced in R2015b

mlreportgen.ppt.Slide.add

Package: mlreportgen.ppt

Add text box, table, or picture to slide

Syntax

```
slideObj = add(slide,object)
```

Description

`slideObj = add(slide,object)` adds a text box, table, or picture to a slide.

Examples

Add a Picture to a Slide

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'mySlideAddPresentation.pptx';
slides = Presentation(slidesFile);
slide1 = add(slides,'Blank');
```

Create an `mlreportgen.ppt.Picture` object.

```
plane = Picture(which('b747.jpg'));
plane.X = '4in';
plane.Y = '4in';
plane.Width = '5in';
plane.Height = '2in';
```

Add the plane picture to `slide1`.

```
add(slide1,plane);
```

Generate the presentation. Open `mySlideAddPresentation.pptx`.

```
close(slides);
```

Open the presentation `mySlideAdd.pptx` file. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```

- “Add and Replace Presentation Content” on page 14-74

Input Arguments

`slide` — Slide to add content to

`mlreportgen.ppt.Slide` object

Slide to add content to, specified as an `mlreportgen.ppt.Slide` object.

`object` — Object to add to slide

PPT object

Object to add to a slide, specified as one of these PPT objects:

- `mlreportgen.dom.TextBox`
- `mlreportgen.dom.Table`
- `mlreportgen.dom.Picture`

Output Arguments

`slideObj` — Slide
`mlreportgen.ppt.Slide` object

Slide, returned as an `mlreportgen.ppt.Slide` object.

See Also

mlreportgen.ppt.Slide

Introduced in R2015b

mlreportgen.ppt.Slide.find

Package: mlreportgen.ppt

Search in slide

Syntax

```
searchResults = find(slide,objectName)
```

Description

`searchResults = find(slide,objectName)` searches for a slide content object whose `Name` property has the value that you specify using `objectName`.

Examples

Search in Slide

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'mySlideFindPresentation.pptx';
slides = Presentation(slidesFile);
slide1 = add(slides,'Title and Content');
```

Search in `slide1` for content object whose `Name` property is `Content`.

```
contents = find(slide1,'Content');
```

Make the content in the `Content` object in `slide1` bold.

```
contents(1).Bold = true;
add(contents(1),'This is in bold');
```

Generate the presentation. Open `mySlideFindPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
close(slides);
```

```
if ispc
    winopen(slidesFile);
end
```

- “Add and Replace Presentation Content” on page 14-74

Input Arguments

`slide` — Slide to search

`mlreportgen.ppt.Slide` object | string

Slide to search, specified as an `mlreportgen.ppt.Slide` object or as a string that contains the name of a slide in the presentation.

`objectName` — Slide content object whose `Name` property to search in

string

Object `Name` property to search for, specified as a string. The search looks for content objects in the specified slide that have a `Name` property that matches the search string. The search looks in the specified slide. Search strings are case sensitive. The search treats the string as a full string.

Output Arguments

`searchResults` — Search results

array

Search results, represented by an array of PPT placeholder objects. The array can contain these placeholder objects:

- `mlreportgen.ppt.ContentPlaceholder`
- `mlreportgen.ppt.TextBoxPlaceholder`
- `mlreportgen.ppt.TablePlaceholder`
- `mlreportgen.ppt.PicturePlaceholder`

See Also

mlreportgen.ppt.Presentation.find | mlreportgen.ppt.Slide

Introduced in R2015b

mlreportgen.ppt.Slide.replace

Package: mlreportgen.ppt

Replace paragraphs, tables, or pictures in slide

Syntax

```
replace(slide,objectName,content)
```

Description

`replace(slide,objectName,content)` replaces existing content of a placeholder with one or more paragraphs, a table, or a picture that you specify. If the type of content you specify in the content input argument is not valid for replacing the placeholder, the `replace` method has no effect.

Examples

Replace Slide Content

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'mySlideReplacePresentation.pptx';
slides = Presentation(slidesFile);
slide1 = add(slides,'Title Slide');
slide2 = add(slides,'Title and Content');
```

Replace the titles in `slide1` and `slide2`.

```
replace(slide1,'Title','Slide Title of Slide 1');
replace(slide2,'Title','Slide Title of Slide 2');
```

Generate the presentation. Open `mySlideReplacePresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
close(slides);

if ispc
    winopen(slidesFile);
end
```

- "Add and Replace Presentation Content" on page 14-74

Input Arguments

`slide` — Slide in which to replace content

`mlreportgen.ppt.Slide` object

Slide in which to replace content, specified as an `mlreportgen.ppt.Slide` object.

`objectName` — Name of placeholder to replace

string

Name of placeholder to replace, specified as a string. The placeholder name must be defined in the template.

`content` — Content to use as replacement

string | `mlreportgen.ppt.Paragaph` object | cell array of strings or `Paragraph` objects, or a combination of both | `mlreportgen.ppt.Table` object | `mlreportgen.ppt.Picture` object

Content to use as replacement, specified as one of these:

- string
- `mlreportgen.ppt.Paragaph` object
- cell array of strings or `Paragraph` objects, or a combination of both
- `mlreportgen.ppt.Table` object
- `mlreportgen.ppt.Picture` object

See Also

mlreportgen.ppt.Slide

Introduced in R2015b

mlreportgen.ppt.Table.append

Package: mlreportgen.ppt

Append row to table

Syntax

```
tableObj = append(table,row)
```

Description

`tableObj = append(table,row)` appends a row to a table.

Examples

Create a Table with Table Rows

Create a presentation.

```
import mlreportgen.ppt.*

slides = Presentation('myTableEntryPresentation.pptx');
add(slides,'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(slides,'Content');
```

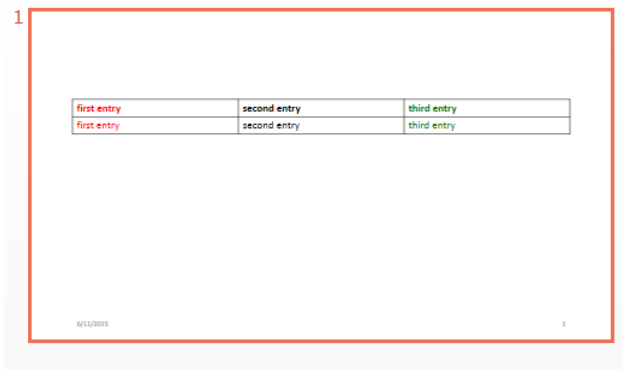
Replace the table in the second slide with `table1`.

```
replace(contents(1),table1);
```

Generate the presentation. Open the `myTableEntryPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
close(slides);

if ispc
    winopen(slidesFile);
end
```



- “Create and Format Tables” on page 14-87

Input Arguments

`table` — Table to append content to

`mlreportgen.ppt.Table` object

Table to append content 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

`tableObj` — Table
`mlreportgen.dom.Table` object

Table, returned as an `mlreportgen.dom.Table` object.

See Also

mlreportgen.ppt.Table | mlreportgen.ppt.TableEntry | mlreportgen.ppt.TableRow

Introduced in R2015b

mlreportgen.ppt.Table.entry

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

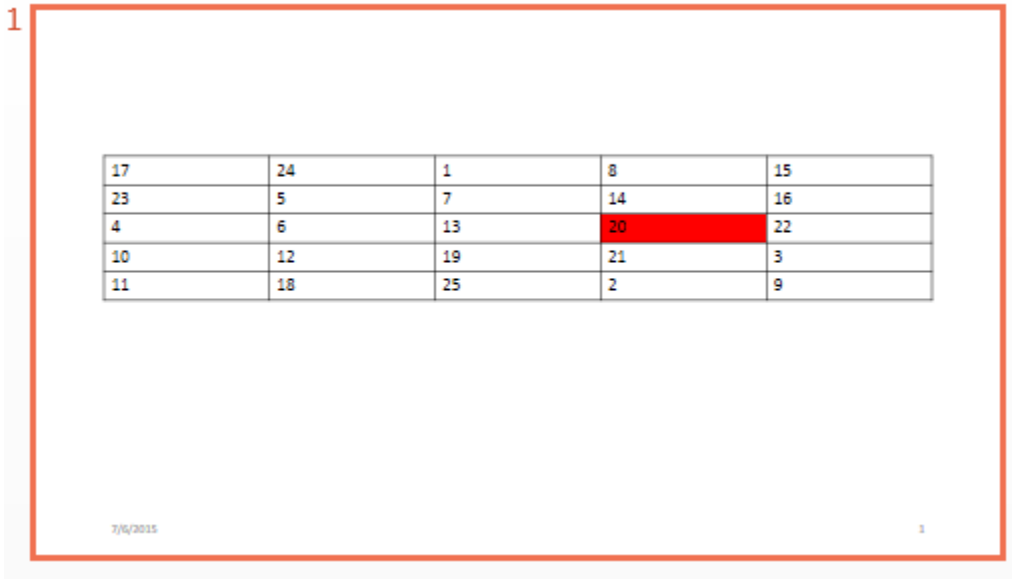
Color a Table Entry

Color the table entry in row 3, column 4.

```
import mlreportgen.ppt.*;
slidesFile = 'myTableEntryMethod.pptx';
slides = Presentation(slidesFile);
slide1 = add(slides,'Title and Content');
t = Table(magic(5));
entry4row3 = t.entry(3,4);
entry4row3.BackgroundColor = 'red';

replace(slide1,'Content',t);

close(slides);
if ispc
winopen(slides.OutputPath);
end
```



- "Create and Format Tables" on page 14-87

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.Table.row | mlreportgen.ppt.TableEntry

Introduced in R2014b

mlreportgen.ppt.Table.replace

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.*

slidesFile = 'myTableReplacePresentation.pptx';
slides = Presentation(slidesFile);
slide1 = add(slides,'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(slides);

if ispc
    winopen(slidesFile);
end
```

- "Add or Replace a Table" on page 14-78
- "Create and Format Tables" on page 14-87

Input Arguments

`table` — Table to replace with another table

`mlreportgen.ppt.Table` object

Table to replace table, specified as an `mlreportgen.ppt.TablePlaceholder` 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.dom.Table` object

Table, returned as an `mlreportgen.dom.Table` object.

See Also

mlreportgen.ppt.Table | mlreportgen.ppt.TablePlaceholder

Introduced in R2015b

mlreportgen.ppt.Table.row

Package: mlreportgen.ppt

Access table row

Syntax

```
tableRowOut = entry(tableObj,row)
```

Description

`tableRowOut = entry(tableObj,row)` returns the specified table row.

Examples

Color a Table Row

Color the third row in the table.

```
import mlreportgen.ppt.*;
slidesFile = 'myTableRowMethod.pptx';
slides = Presentation(slidesFile);
slide1 = add(slides,'Title and Content');
t = Table(magic(5));
row3 = t.row(3);
row3.BackgroundColor = 'red';

replace(slide1,'Content',t);

close(slides);
if ispc
winopen(slides.OutputPath);
end
```

- “Create and Format Tables” on page 14-87

Input Arguments

`tableObj` — Table containing row

`mlreportgen.ppt.Table` object

Table containing the row, specified as an `mlreportgen.ppt.Table` object.

`row` — 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.Table.entry | mlreportgen.ppt.TableRow

Introduced in R2014b

mlreportgen.ppt.TableEntry.append

Package: mlreportgen.ppt

Append string or paragraph to table entry

Syntax

```
tableEntryObj = append(tableEntry,content)
```

Description

`tableEntryObj = append(tableEntry,content)` appends a string or `Paragraph` object to a table entry.

Examples

Create a Table with Table Rows and Entries

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myTableEntryPresentation.pptx';
slides = Presentation(slidesFile);
add(slides,'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(slides,'Content');
```

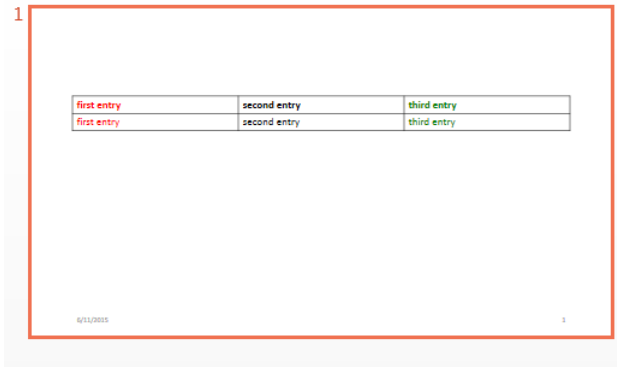
Replace the table in the second slide with `table1`.

```
replace(contents(1),table1);
```

Generate the presentation. Open `myTableEntryPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
close(slides);

if ispc
    winopen(slidesFile);
end
```



- “Add or Replace a Table” on page 14-78
- “Create and Format Tables” on page 14-87

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

string | `mlreportgen.ppt.Paragraph` object

Content to append to a table entry, specified as a string or one or more `mlreportgen.ppt.Paragraph` objects.

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.TableEntry | mlreportgen.ppt.TableRow

Introduced in R2015b

mlreportgen.ppt.TablePlaceholder.replace

Package: mlreportgen.ppt

Replace table in table placeholder

Syntax

```
tableObj = replace(tablePlaceholder,table)
```

Description

`tableObj = replace(tablePlaceholder,table)` replaces the table in a table placeholder.

Examples

Replace Table in Table Placeholder

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myTableReplacePresentation.pptx';
slides = Presentation(slidesFile);
slide1 = add(slides,'Title and Table');
```

Create an `mlreportgen.ppt.Table` object.

```
t1 = Table(magic(7));
```

Search in `slide1` for `Table`.

```
contents = find(slide1,'Table');
```

Replace the first table placeholder whose `Name` property is `Table` with a table.

```
replace(contents(1),t1);
```

Generate the presentation. Open `myTableReplacePresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
close(slides);

if ispc
    winopen(slidesFile);
end
```

- "Access PowerPoint Template Elements" on page 14-37
- "Add or Replace a Table" on page 14-78

Input Arguments

`tablePlaceholder` — Table placeholder to replace table

`mlreportgen.ppt.TablePlaceholder` object

Table placeholder to replace table, specified as an `mlreportgen.ppt.TablePlaceholder` object.

`table` — Table to use as replacement

`mlreportgen.ppt.Table` object

Table to use as a 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

Introduced in R2015b

mlreportgen.ppt.TableRow.append

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.*

slidesFile = 'myTableEntryPresentation.pptx';
slides = Presentation(slidesFile);
add(slides,'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(slides,'Content');
```

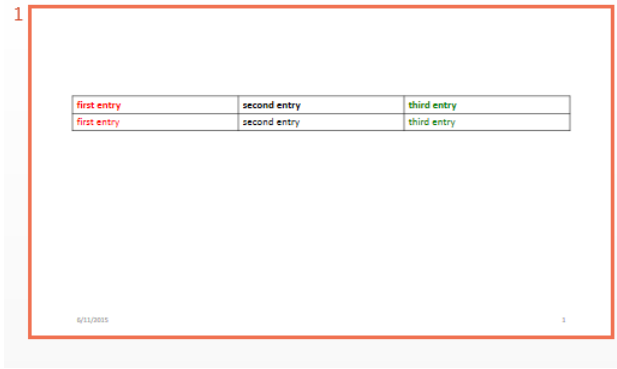
Replace the table in the second slide with `table1`.

```
replace(contents(1),table1);
```

Generate the presentation. Open `myTableEntryPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
close(slides);

if ispc
    winopen(slidesFile);
end
```



- “Add or Replace a Table” on page 14-78
- “Create and Format Tables” on page 14-87

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.TableEntry | mlreportgen.ppt.TableRow

Introduced in R2015b

mlreportgen.ppt.TextBox.add

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 placeholder.

Examples

Add Text to Text Box

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myTextBoxAddPresentation.pptx';
slides = Presentation(slidesFile);
slide1 = add(slides,'Blank');
```

Add a text box to the blank slide (`slide1`).

```
tb = TextBox();
tb.X = '1in';
tb.Y = '1in';
tb.Width = '4 in';
tb.Height = '2in';

add(slide1,tb);
```

Add text to the text box.

```
para = add(tb,'This is the content');
para.Bold = true;
```

Generate the presentation.

```
close(slides);
```


- "Add and Replace Text" on page 14-75
- "Create and Format Text" on page 14-81

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

string | array | cell array | `mlreportgen.ppt.Paragraph`

Content to add to a text box, specified as a string, array, cell array, or `Paragraph` object. If you specify an array, include only `mlreportgen.ppt.Paragraph` objects in the array. If you specify a cell array, you can include strings and `mlreportgen.ppt.Paragraph` objects.

`contents` — Multiple paragraphs to add

cell array of strings or `mlreportgen.ppt.Paragraph` objects or a combination of both | cell array of `mlreportgen.ppt.Paragraph` objects | cell array of a combination of strings and `mlreportgen.ppt.Paragraph` objects

Content to add, specified as a cell array of strings, `mlreportgen.ppt.Paragraph` objects, or a combination of both. Inner cell arrays specify inner list items. The slide layout specifies whether the text displays as paragraphs, bulleted list items, or numbered list items.

Output Arguments

`paraObj` — Paragraph

`mlreportgen.dom.Paragraph` object

Paragraph, returned as an `mlreportgen.dom.Paragraph` object.

See Also

mlreportgen.ppt.Paragraph | mlreportgen.ppt.TextBox | mlreportgen.ppt.TextBox.replace

Introduced in R2015b

mlreportgen.ppt.TextBox.replace

Package: mlreportgen.ppt

Replace text box paragraphs

Syntax

```
paraObj = replace(textBox,content)
replace(textBox,contents)
```

Description

`paraObj = replace(textBox,content)` replaces 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.*

slidesFile = 'myTextBoxReplacePresentation.pptx';
slides = Presentation(slidesFile);
slide1 = add(slides,'Blank');
```

Create an `mlreportgen.ppt.Paragraph` object.

```
p = Paragraph('Hello World');
```

Add a text box to the blank slide (`slide1`).

```
tb = TextBox();
tb.X = '1in';
tb.Y = '1in';
```

```
tb.Width = '4 in';
tb.Height = '2in';

add(slide1,tb);
```

Add replace the text box with the paragraph.

```
add(tb,p);
```

Replace the content of the text box.

```
replace(tb,'This is the real content');
```

Generate the presentation.

```
close(slides);
```

- "Add and Replace Text" on page 14-75
- "Create and Format Text" on page 14-81

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

string | array | cell array

Text to use as replacement, specified as a string, array, cell array, or `Paragraph` object. If you specify an array, include only `mlreportgen.ppt.Paragraph` objects in the array. If you specify a cell array, you can include strings and `mlreportgen.ppt.Paragraph` objects.

`contents` — Multiple paragraphs to use as replacement

cell array of strings or `mlreportgen.ppt.Paragraph` objects or a combination of both | cell array of `mlreportgen.ppt.Paragraph` objects | cell array of a combination of strings and `mlreportgen.ppt.Paragraph` objects

Content to use as a replacement, specified as a cell array of strings, `mlreportgen.ppt.Paragraph` objects, or a combination of both. Inner cell arrays

specify inner list items. The slide layout specifies whether the text displays as paragraphs, bulleted list items, or numbered list items.

Output Arguments

`paraObj` — Paragraph

`mlreportgen.dom.Paragraph` object

Paragraph, returned as an `mlreportgen.dom.Paragraph` object.

See Also

mlreportgen.ppt.Paragraph | mlreportgen.ppt.TextBox | mlreportgen.ppt.TextBox.add

Introduced in R2015b

mlreportgen.ppt.TextBoxPlaceholder.add

Package: mlreportgen.ppt

Add paragraphs to text box placeholder

Syntax

```
paraObj = add(textBoxPlaceholder,content)
add(textBoxPlaceholder,contents)
```

Description

`paraObj = add(textBoxPlaceholder,content)` adds text in a text box placeholder.

`add(textBoxPlaceholder,contents)` adds multiple paragraphs to a text box placeholder.

Examples

Add Text to Text Box Placeholder

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myTextBoxPlaceholderAddPresentation.pptx';
slides = Presentation(slidesFile);
add(slides,'Title and Content');
```

Create an `mlreportgen.ppt.Paragraph` object.

```
p = Paragraph('Hello World');
```

Use the `Presentation.find` method to find content the text box placeholder called `Title` in the presentation. Replace the title for the first slide with the paragraph.

```
contents = find(slides,'Title');
```

Add the paragraph to the first slide.

```
replace(contents(1),p);
```

Add the paragraph to the first slide.

```
add(contents(1),' -- How are You?');
```

Generate the presentation.

```
close(slides);
```


- "Access PowerPoint Template Elements" on page 14-37
- "Add and Replace Text" on page 14-75

Input Arguments

`textBoxPlaceholder` — Text box placeholder to add text to

`mlreportgen.ppt.TextBoxPlaceholder` object

Text box placeholder to add text to, specified as an `mlreportgen.ppt.TextBoxPlaceholder` object.

`content` — Text to add to text box placeholder

string | array | cell array | `mlreportgen.ppt.Paragraph`

Content to add to a text box placeholder, specified as a string, array, cell array, or `Paragraph` object. If you specify an array, include only `mlreportgen.ppt.Paragraph` objects in the array. If you specify a cell array, you can include strings and `mlreportgen.ppt.Paragraph` objects.

`contents` — Multiple paragraphs to add

cell array of strings or `mlreportgen.ppt.Paragraph` objects or a combination of both | cell array of `mlreportgen.ppt.Paragraph` objects | cell array of a combination of strings and `mlreportgen.ppt.Paragraph` objects

Text to add, specified as a cell array of strings, `mlreportgen.ppt.Paragraph` objects, or a combination of both. Inner cell arrays specify inner list items. The slide layout specifies whether the text displays as paragraphs, bulleted list items, or numbered list items.

Output Arguments

`paraObj` — Paragraph
`mlreportgen.dom.Paragraph` object

Paragraph, returned as an `mlreportgen.dom.Paragraaph` object.

See Also

mlreportgen.ppt.Paragraph | mlreportgen.ppt.TextBoxPlaceholder | mlreportgen.ppt.TextBoxPlaceholder.replace

Introduced in R2015b

mlreportgen.ppt.TextBoxPlaceholder.replace

Package: mlreportgen.ppt

Replace text box placeholder paragraphs

Syntax

```
paraObj = replace(textBoxPlaceholder,content)
replace(textBoxPlaceholder,contents)
```

Description

`paraObj = replace(textBoxPlaceholder,content)` replaces the text in a text box placeholder.

`replace(textBoxPlaceholder,contents)` replaces multiple paragraphs in a text box placeholder.

Examples

Replace Text in a Text Box Placeholder

Create a presentation.

```
import mlreportgen.ppt.*

slides = Presentation('myTextBoxReplacePresentation');
add(slides,'Title and Content');
```

Create an `mlreportgen.ppt.Paragraph` object.

```
p = Paragraph('Hello World');
```

Use the `Presentation.find` method to find text box placeholders called `Title` in the presentation. Replace the title for the first slide with the paragraph.

```
contents = find(slides,'Title');
```

Add the paragraph to the first slide.

```
replace(contents(1),p);
```

Close the presentation.

```
close(slides);
```


- "Access PowerPoint Template Elements" on page 14-37
- "Add and Replace Text" on page 14-75

Input Arguments

`textBoxPlaceholder` — Text box placeholder

`mlreportgen.ppt.TextBoxPlaceholder` object

Text box placeholder to replace text in, specified as an `mlreportgen.ppt.TextBoxPlaceholder` object.

`content` — Text to use as replacement

string | array | cell array

Text to use as replacement, specified as a string, array, cell array, or `Paragraph` object. If you specify an array, include only `mlreportgen.ppt.Paragraph` objects in the array. If you specify a cell array, you can include strings and `mlreportgen.ppt.Paragraph` objects.

`contents` — Multiple paragraphs to use as replacement

cell array of strings or `mlreportgen.ppt.Paragraph` objects or a combination of both | cell array of `mlreportgen.ppt.Paragraph` objects | cell array of a combination of strings and `mlreportgen.ppt.Paragraph` objects

Content to use as a replacement, specified as a cell array of strings, `mlreportgen.ppt.Paragraph` objects, or a combination of both. Inner cell arrays specify inner list items. The slide layout specifies whether the text displays as paragraphs, bulleted list items, or numbered list items.

Output Arguments

`paraObj` — Paragraph

`mlreportgen.ppt.Paragraph` object

Paragraph, represented by an `mlreportgen.ppt.Paragraph` object.

See Also

mlreportgen.ppt.Paragraph | mlreportgen.ppt.TextBoxPlaceholder | mlreportgen.ppt.TextBoxPlaceholder.add

Introduced in R2015b

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 PPT file in PowerPoint on Windows or Apache OpenOffice on Linux.

`[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 in PowerPoint or OpenOffice

Open a PPT file. The file opens in PowerPoint on Windows or Apache OpenOffice on Linux.

```
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
string

PPT file to open or convert, specified as a string 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
string

Error or warning information about opening or converting the file, returned as a string.

See Also
`rptconvert | rptview`

Introduced in R2016a

report

Generate reports from report setup file

Syntax

```
report
report (filename,...)
report (___,-oOPATH)
report (___,-fFORMAT)
report (___,-genOption1,...)
[report1, report2, ...] = report (rptfile1, rptfile2, ...)
```

Description

- `report` with no arguments opens the Report Explorer. For more information on the Report Explorer, see "Working with the Report Explorer" on page 1-8
- `report (filename,...)` generates a report from the specified report setup files. You can specify one or more report setup files. When specifying the name of the report setup file, omit the `.rpt` file name extension.
- `report (___,-oOPATH)` sets the name of the generated report. You can specify a path or a single file name for the `OPATH` path argument.
- `report (___,-fFORMAT)` sets the output format and file name extension of the generated report. Supported formats include:

  - Adobe Acrobat PDF (`.pdf`)
  - HTML (`.html`)
  - Microsoft Word (`.doc`)
  - Rich Text format (`.rtf`)

  For example, `report('simple-report','-fPDF')` generates a PDF file.
- `report (___,-genOption1,...)` specifies one or more of the following report generation options:

  - `-noview` — Prevents launching the file viewer

  - `-graphical` — Shows hierarchy in Report Explorer
  - `-debug` — Enables debug mode
  - `-quiet` — Sets error echo level to `0`
  - `-sSTYLESHEETNAME` — Sets stylesheet name (not required when choosing format)
- `[report1, report2, ...] = report (rptfile1, rptfile2, ...)` returns the names of the generated reports. If the MATLAB Report Generator software cannot generate a given report, its returned name is empty.

Note: For reports that use the `Word Document` format, you must have Microsoft Word software installed on the machine that you use to generate the report.

Examples

Example 1: Setting the format of the generated report

- 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
  ```

Example 2: Specifying the file and path of the generated report

Generate a report named `simple-report` in the folder `/tmp/index.html`:

```
report ('simple-report','-o/tmp/index.html')
```

More About

- “Generate Reports”

See Also

`setedit` | `rptconvert` | `rptlist` | `compwiz`

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 stylesheet. If you omit this argument, the default stylesheet 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 stylesheet identifiers. The second column includes descriptions of each stylesheet.
- `sheetlist = rptconvert('-stylesheetlist',format)` returns an array like that returned by `sheetlist = rptconvert('-stylesheetlist')`. The first column of this array includes stylesheet 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` stylesheets:

```
rptconvert('-stylesheetlist','html')
```

More About

- “Convert XML Documents to Different File Formats” on page 4-18

See Also

`setedit` | `report` | `rptlist` | `compwiz`

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`

rptview

Display DOM report

Syntax

```
rptview(reportPath)
rptview(reportPath,'pdf')

rptview(reportName,format)
```

Description

`rptview(reportPath)` displays the report `reportPath` in an appropriate viewer, based on the file extension. You can use the `Document.OutputPath` property to specify `reportPath`.

`rptview(reportPath,'pdf')` converts a Word report to PDF and displays the report in a PDF viewer.

`rptview(reportName,format)` displays the report `reportPath` in an appropriate viewer, based on the format specified in `format`.

Examples

Display Report in HTML Viewer

Display an HTML report. Include the file extension when you specify the report name in the `rptview` function.

```
import mlreportgen.dom.*;
d = Document('mydoc','html');

p = Paragraph('Hello World');
append(d,p);

close(d);
```

```
rptview('mydoc.htmx');
```

Convert a Word Report and Display It in a PDF Viewer

Use the `rptview` function to convert a Word report to PDF and display it in a PDF viewer.

```
import mlreportgen.dom.*;
d = Document('mydoc','docx');

p = Paragraph('Hello World');
append(d,p);

close(d);
rptview('mydoc.docx','pdf');
```

Display Report Using the OutputPath Property

Display a report using the value of the `OutputPath` property of the `mlreportgen.dom.Document` object of the report.

```
import mlreportgen.dom.*;
d = Document('mydoc','docx');

p = Paragraph('Hello World');
append(d,p);

close(d);
rptview(d.OutputPath);
```

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.*;
d = Document('mydoc','html');

p = Paragraph('Hello World');
append(d,p);
close(d);

dWord = Document('mydoc','docx');
p = Paragraph('Hello again, World');
append(dWord,p);
```

```
close(dWord);

rptview('mydoc','docx');
```

Input Arguments

`reportPath` — Report file path including file extension

string

Path to a specific report file, including the file extension, specified as a string.

The report file name extension determines the viewer in which the report displays.

File Extension	Viewer
`.htmx`	MATLAB Web browser
`.zip`	MATLAB Web browser
`.docx`	The report displays in Microsoft Word unless you add the `'pdf'` argument after `reportPath`.

`reportName` — Report name

string

The full path of a report, without the file extension, specified as a string. You can specify a string with the full path. Alternatively, you can use the value of the `OutputPath` property of the `mlreportgen.dom.Document` object that you create for the report.

`format` — Report output format

string

Use one of these values:

- `'html'`
- `'docx'`
- `'pdf'`

See Also

mlreportgen.dom.Document

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.

More About

- "Working with the Report Explorer" on page 1-8

See Also

`rptlist` | `report` | `rptconvert`

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.htmtx','H:\report_templates\myTemplate');
```

Input Arguments

`zippedTemplatePath` — Path of the zipped DOM template

string

If you do not include a file extension in the path, the function assumes the extension is `.htmtx`.

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
string

The template is unzipped into the folder that you specify in `unzippedTemplate` path.

More About

- "Report Packages" on page 13-17

See Also

mlreportgen.dom.Document.createTemplate | `zipTemplate`

Introduced in R2014b

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 `htmtx` or `pdftx`. 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.htmtx`.

```
zipTemplate('myTemplate');
```

Zip to a Specified Zip File Name

Zip the template `myTemplate` into a zip file called `myReportTemplate.htmtx`.

```
zipTemplate('myReportTemplate.htmtx','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

string

Path to folder containing unzipped template, specified as a string. The file names in the unzipped template must contain only ASCII characters.

`zippedTemplate` — Path for zipped DOM template file

string

Full path for the zipped DOM template, including the file extension `.htmtx` or `.pdftx`, specified as a string.

`mainDocument` — Name of main document file

string

Main document file name, including the file extension, specified as a string.

`partTemplates` — Document part library file name

string

Document part library file name, including the file extension, specified as a string.

More About

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`.

- "Report Packages" on page 13-17

See Also

mlreportgen.dom.Document.createTemplate | `unzipTemplate`

Introduced in R2014b

Classes – Alphabetical List

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`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.RepeatAsHeaderRow | mlreportgen.dom.TableRow

More About

- “Report Formatting Approaches” on page 13-21

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

string

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

string

Name of number style defined in the template, specified as a string. 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

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`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

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`CustomAttributes` — Custom attributes of document element

array of `mlreportgen.dom.CustomAttribute` objects

The output format must support the custom attributes of this document element.

`FontFamilyName` — Name of font family for text

string

The name of a font family.

To specify substitutions for this font, do not set this property. Instead create and add a `mlreportgen.dom.FontFamily` object to the `Style` property of this document element.

Setting the `FontFamilyName` property adds a corresponding `mlreportGen.dom.FontFamily` format object to the `Style` property for this document element. Removing the `FontFamilyName` property setting removes the object.

`FontSize` — Font size

string

If you need to specify substitutions for this font, do not set this property. Instead create and add a `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.

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the font size is expressed. Use one of these abbreviations for the units for the font size.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixel

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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

string

The default for this property is []. You can set it to one of these values:

- `none` — Do not use strikethrough.
- `single` — Use a single line for strikethrough.
- `double` — Use a double line for strikethrough for Word documents.

Setting the `Strike` property adds a corresponding `mlreportGen.dom.Strike` format object to the `Style` property for this document element. Removing the `Strike` property setting 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 stylesheet style specified by the `StyleName` property of this element. Formats that do not apply to this element are ignored.

`StyleName` — Style for the number

string

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 document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Underline` — Type of underline, if any, for text

[ ] (default) | string

You can specify one of the following types of underlines.

Border String	Description	Supported Output Types
`dash`	Dashed underline	Word
`dashedHeavy`	Line with heavy dashes	Word
`dashLong`	Line with long dashes	Word
`dashLongHeavy`	Line with heavy long dashes	Word
`dashDotDotHeavy`	Line with heavy dashes with two dots between the dashes	Word
`dashDotHeavy`	Heavy dash-dot line	Word
`dotted`	Dotted line	Word
`dottedHeavy`	Thick dotted line	Word
`dotDash`	Dot-dash line	Word
`dotDotDash`	Dot-dot-dash line	Word
`dashDotHeavy`	Heavy dot-dash line	Word
`double`	Double line	Word
`none`	Do not use underlining	All output types
`single`	Single line	All output types
`thick`	Thick line	Word
`wave`	Wavy line	Word

Border String	Description	Supported Output Types
`waveyDouble`	Double wavy line	Word
`waveyHeavy`	Heavy wavy	Word
`words`	Underline non-space characters only	Word

If this property is empty and `StyleName` property of this document element specifies a style sheet style, the type of underline is determined by that style.

To specify the color as well as the type of the underline, do not set the `Underline` property. Instead, set the `Style` property of this document element to include an `mlreportgen.dom.Underline` format object that specifies the desired underline type and color.

Setting the `Underline` property adds a corresponding `mlreportgen.dom.Underline` format object to the `Style` property for this document element. Removing the `Underline` property setting removes the object.

`WhiteSpace` — White space and line breaks in text

[ ] (default) | string

To specify how to handle white space, use one of these values.

String	Description	Supported Output Types
`normal` (default)	For HTML and PDF, removes spaces at beginning and end of text string. Multiple spaces within string collapse to single space. For Word, removes spaces at beginning and end of text string.	All output types
`nowrap`	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML

String	Description	Supported Output Types
`preserve`	Preserves spaces and line feeds. Acts like the `<pre>` tag in HTML.	All output types
`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.	HTML and PDF
`pre-wrap`	Preserves white space. Text wraps when necessary and on line breaks	HTML and PDF

Setting the `WhiteSpace` property adds a corresponding `WhiteSpace` format object to `Style` property. Removing the `WhiteSpace` property setting removes the `WhiteSpace` object.

Methods

Method	Purpose
append Use `AutoNumber.append` in a similar way to how you use `ExternalLink.append`.	Append a custom element to this number.
clone Use `AutoNumber.clone` in a similar way to how you use `Paragraph.clone`.	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);
```

- “Automatically Number Document Content” on page 13-98

See Also

mlreportgen.dom.CounterInc | mlreportgen.dom.CounterReset | mlreportgen.dom.Document.createAutoNumberStream | mlreportgen.dom.Document.getAutoNumberStream

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 `mlreportgen.dom.Document.createAutoNumberStream` method.

Properties

`CharacterCase` — Character case of generated numbers

string

Values can be:

- `lower` — lowercase (for example, a,b,c)
- `lower` — uppercase (for example, A,B,C)

`CharacterType` — Character type of generated numbers

string

Values can be:

- `alphabetic` — for example, a,b,c
- `arabic` — for example, 1,2,3
- `roman` — for example, i,ii,iii

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`InitialValue` — Initial value of generated number

string

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

string

Name of numbering stream, specified as a string.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

See Also

mlreportgen.dom.CounterInc | mlreportgen.dom.CounterReset | mlreportgen.dom.Document.createAutoNumberStream | mlreportgen.dom.Document.getAutoNumberStream

Related Examples

- “Automatically Number Document Content” on page 13-98

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 hexidecimal RGB color value.

Input Arguments

`colorName` — Name of a color to use

string

The name of a color, specified as a string. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.

`rgbValue` — Hexidecimal RGB (red, green, blue) color value

string

A string using the following RGB format: `#RRGGBB`. Use # as the first character and two-digit hexidecimal numbers for 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)

string

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — CSS color name or hexadecimal RGB value for this color

string

Either a CSS color name or a hexidecimal RGB value, specified as a string.

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 could 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

More About

- “Report Formatting Approaches” on page 13-21

mlreportgen.dom.Bold class

Package: mlreportgen.dom

Bold for text object

Description

Specifies whether to use bold for a text object

Construction

`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`

Output Arguments

`boldObj` — Bold text

`mlreportgen.dom.Bold` object

Bold text, represented by an `mlreportgen.dom.Bold` object

Properties

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`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

More About

- “Report Formatting Approaches” on page 13-21

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

string

Use one of these values.

String	Applies To	
	DOCX	HTML and PDF
`'dashed'`	✓	✓
`'dashdotstroked'`	✓	
`'dashsmallgap'`	✓	
`'dotted'`	✓	✓

String	Applies To	
	DOCX	HTML and PDF
'dotdash'	✓	
'dotdotdash'	✓	
'double'	✓	✓
'doublewave'	✓	
'inset'	✓	✓
'none'	✓	✓
'outset'	✓	✓
'single'	✓	
'solid'		✓
'thick'	✓	
'thickthinlargegap'	✓	
'thickthinmediumgap'	✓	
'thickthinsmallgap'	✓	
'thinthicklargegap'	✓	
'thinthickmediumgap'	✓	
'thinthicksmallgap'	✓	
'thinthickthinlargegap'	✓	
'thinthickthinmediumgap'	✓	

String	Applies To	
	DOCX	HTML and PDF
`'thinthickthinsmallgap'`	✓	
`'threedemboss'`	✓	
`'threedengrave'`	✓	
`'triple'`	✓	
`'wave'`	✓	

`color` — Color of border

string

You can specify:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`width` — Width of border

string

String specifying the width of the border. String 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
- `pi` — picas
- `pt` — points
- `px` — pixel

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

string

You can specify:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Style` — Default style of border segments

string

For details, see the description of the `style` input argument for the `mlreportgen.dom.Border` constructor.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Width` — Width of border

string

String specifying the width of the border. String 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
- `pi` — picas
- `pt` — points
- `px` — pixel

`BottomColor` — Bottom border segment color

string

Bottom border segment color, specified as a string.

`BottomStyle` — Bottom border segment style

string

Bottom border segment style, specified as a string.

`BottomWidth` — Bottom border segment width

string

Bottom border segment width, specified as a string.

`TopColor` — Top border segment color

string

Top border segment color, specified as a string.

`TopStyle` — Top border segment style

string

Top border segment style, specified as a string.

`TopWidth` — Top border segment width

string

Top border segment width, specified as a string.

`LeftColor` — Left border segment color

string

Left border segment color, specified as a string.

`LeftStyle` — Left border segment style

string

Left border segment style, specified as a string.

`LeftWidth` — Left border segment width

string

Left border segment width, specified as a string.

`RightColor` — Right border segment color

string

Right border segment color, specified as a string.

`RightStyle` — Right border segment style

string

Right border segment style, specified as a string.

`RightWidth` — Right border segment width

string

Right border segment width, specified as a string.

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);
```

- “Create and Format Tables” on page 13-69

See Also

mlreportgen.dom.ColSep | mlreportgen.dom.RowSep | mlreportgen.dom.Table

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

string

String specifying to collapse table borders (`'on'`) or to leave table and adjacent cell borders separate (`'off'`).

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Specify whether to collapse border

string

String specifying to collapse table borders (`'on'`) or to leave table and adjacent cell borders separate (`'off'`).

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);
```

- “Create and Format Tables” on page 13-69

See Also

mlreportgen.dom.Border | mlreportgen.dom.TableColSpec | mlreportgen.dom.TableEntry | mlreportgen.dom.TableRow

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 string of `n` special characters.

Input Arguments

`name` — Specify character entity name

string

String must be a name that is listed at http://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

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`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 `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

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`Content` — Text string contained by this document element

string

Text string contained by this document element.

`CustomAttributes` — Custom attributes of document element

array of `mlreportgen.dom.CustomAttribute` objects

The output format must support the custom attributes of this document element.

`FontFamilyName` — Name of font family for text

string

The name of a font family.

To specify substitutions for this font, do not set this property. Instead create and add a `mlreportgen.dom.FontFamily` object to the `Style` property of this document element.

Setting the `FontFamilyName` property adds a corresponding `mlreportGen.dom.FontFamily` format object to the `Style` property for this document element. Removing the `FontFamilyName` property setting removes the object.

`FontSize` — Font size

string

If you need to specify substitutions for this font, do not set this property. Instead create and add a `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.

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the font size is expressed. Use one of these abbreviations for the units for the font size.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixel

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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

string

The name is a character entity listed in http://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

string

The default for this property is []. You can set it to one of these values:

- `none` — Do not use strikethrough.
- `single` — Use a single line for strikethrough.
- `double` — Use a double line for strikethrough for Word documents.

Setting the `Strike` property adds a corresponding `mlreportGen.dom.Strike` format object to the `Style` property for this document element. Removing the `Strike` property setting 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

string

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 document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Underline` — Type of underline, if any, for text

`[ ]` (default) | string

You can specify one of the following types of underlines.

Border String	Description	Supported Output Types
`dash`	Dashed underline	Word
`dashedHeavy`	Line with heavy dashes	Word
`dashLong`	Line with long dashes	Word
`dashLongHeavy`	Line with heavy long dashes	Word
`dashDotDotHeavy`	Line with heavy dashes with two dots between the dashes	Word
`dashDotHeavy`	Heavy dash-dot line	Word
`dotted`	Dotted line	Word

Border String	Description	Supported Output Types
`dottedHeavy`	Thick dotted line	Word
`dotDash`	Dot-dash line	Word
`dotDotDash`	Dot-dot-dash line	Word
`dashDotHeavy`	Heavy dot-dash line	Word
`double`	Double line	Word
`none`	Do not use underlining	All output types
`single`	Single line	All output types
`thick`	Thick line	Word
`wave`	Wavy line	Word
`waveyDouble`	Double wavy line	Word
`waveyHeavy`	Heavy wavy	Word
`words`	Underline non-space characters only	Word

If this property is empty and `StyleName` property of this document element specifies a style sheet style, the type of underline is determined by that style.

To specify the color as well as the type of the underline, do not set the `Underline` property. Instead, set the `Style` property of this document element to include an `mlreportgen.dom.Underline` format object that specifies the desired underline type and color.

Setting the `Underline` property adds a corresponding `mlreportgen.dom.Underline` format object to the `Style` property for this document element. Removing the `Underline` property setting removes the object.

`WhiteSpace` — White space and line breaks in text

[ ] (default) | string

To specify how to handle white space, use one of these values.

String	Description	Supported Output Types
`normal` (default)	For HTML and PDF, removes spaces at beginning	All output types

String	Description	Supported Output Types
	and end of text string. Multiple spaces within string collapse to single space. For Word, removes spaces at beginning and end of text string.	
`nowrap`	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
`preserve`	Preserves spaces and line feeds. Acts like the `<pre>` tag in HTML.	All output types
`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.	HTML and PDF
`pre-wrap`	Preserves white space. Text wraps when necessary and on line breaks	HTML and PDF

Setting the `WhiteSpace` property adds a corresponding `WhiteSpace` format object to `Style` property. Removing the `WhiteSpace` property setting removes the `WhiteSpace` object.

Methods

Method	Purpose
append	Append a custom element to this character entity.

Method	Purpose
Use `CharEntity.append` in a similar way to how you use `ExternalLink.append`.	
clone Use `CharEntity.clone` in a similar way to how you use `Paragraph.clone`.	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.Paragraph | mlreportgen.dom.Text

More About

- "Report Formatting Approaches" on page 13-21

mlreportgen.dom.Color class

Package: mlreportgen.dom

Color of document element

Description

Specifies the color of a document element.

Construction

`colorObj = Color()` creates a black color object.

`colorObj = Color(colorName)` creates a color object based on the specified CSS color name.

`colorObj = Color(rgbValue)` creates a color object using the hexidecimal RGB color value.

Input Arguments

`colorName` — Name of color

string

Name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.

`rgbValue` — Hexidecimal RGB (red, green, blue) color value

string

A string using the following RGB format: `#RRGGBB`. Use # as the first character and two-digit hexidecimal numbers for 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` — hexidecimal color value (read-only)

string

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — CSS color name or hexidecimal RGB value for this color

string

Either a CSS color name or a hexidecimal RGB value.

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

More About

- “Report Formatting Approaches” on page 13-21

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

string

String specifying the style of the table column separator.

String	Applies To	
	DOCX	HTML and PDF
`'dashed'`	✓	✓
`'dashdotstroked'`	✓	
`'dashsmallgap'`	✓	
`'dotted'`	✓	✓

String	Applies To	
	DOCX	HTML and PDF
'dotdash'	✓	
'dotdotdash'	✓	
'double'	✓	✓
'doublewave'	✓	
'inset'	✓	✓
'none'	✓	✓
'outset'	✓	✓
'single'	✓	
'solid'		✓
'thick'	✓	
'thickthinlargegap'	✓	
'thickthinmediumgap'	✓	
'thickthinsmallgap'	✓	
'thinthicklargegap'	✓	
'thinthickmediumgap'	✓	
'thinthicksmallgap'	✓	
'thinthickthinlargegap'	✓	
'thinthickthinmediumgap'	✓	

String	Applies To	
	DOCX	HTML and PDF
`'thinthickthinsmallgap'`	✓	
`'threedemboss'`	✓	
`'threedengrave'`	✓	
`'triple'`	✓	
`'wave'`	✓	

`color` — Color of column separator in table

string

You can specify:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`width` — Width of column separator in the table

string

String specifying the color of the table column separator. String 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
- `pi` — picas
- `pt` — points
- `px` — pixel

Output Arguments

`colSepObj` — Column separator definition

`mlreportgen.dom.ColSpec` object

Column separator definition, represented by an `mlreportgen.dom.ColSpec` object.

Properties

`Color` — Separator color

string

You can specify:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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 stylesheet style specified by the `StyleName` property.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Width` — Separator width

string

String representing a percentage (for example, `'100%'`) or a number of units of measurement, having the format `valueUnits`, where `Units` is an abbreviation for the units in which the width is expressed.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the width is expressed. Use one of these abbreviations for the units of a width.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixel

Data Types: `char`

Examples

Specify Table Column Formatting

Set the width and color of the first column of a table.

```
import mlreportgen.dom.*
doc = Document('myTableColReport','docx');

grps(1) = TableColSpecGroup;
grps(1).Span = 1;
grps(1).Style = {Color('red'),Width('1in')};

table = Table(magic(5));
table.ColSpecGroups = grps;
append(doc,table);
```

```
close(doc);
rptview('myTableColReport','docx');
```

- “Create and Format Tables” on page 13-69

See Also

mlreportgen.dom.RowSep

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

string

HTML container tag name, specified as a string. 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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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

string

HTML container tag name, specified as a string. 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

This read-only property lists the parent of this document element.

`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 stylesheet style specified by the `StyleName` property of this element. Formats that do not apply to this element are ignored.

`StyleName` — Style name

string

Style name, specified as a string. The style name is the name of a style specified in the stylesheet 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 document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Methods

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);
```

- “Create Object Containers” on page 13-30
- “Add Content as a Group” on page 13-14

See Also

mlreportgen.dom.Group

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
string

Category of a document, specified as a string.

`ContentStatus` — Content status of document
string

Content status of a document, specified as a string.

`Creator` — Creator of document
string

Creator of a document, specified as a string.

`Description` — Description of document
string

Description of a document, specified as a string.

`Id` — ID for document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Identifier` — Identifier for document
string

Identifier for a document, specified as a string.

`Keywords` — Keywords associated with document
array of strings

Keywords associated with a document, specified as a string.

`Language` — Language of document
string

Language of a document, specified as a string.

`LastModifiedBy` — Agent that last modified this document
string

Agent that last modified this document, specified as a string.

`Revision` — Revision of document
string

Revision of a document, specified as a string.

`Subject` — Subject of document
string

Subject of a document, specified as a string.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Title` — Title of document

string

Title of a document, specified as a string.

`Version` — Version of document

string

Version of a document, specified as a string.

See Also

mlreportgen.dom.Document.getCoreProperties |
mlreportgen.dom.Document.setCoreProperties

More About

- “Report Packages” on page 13-17

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
string

Numbering stream names, specified as a string. To specify multiple stream names, add a space between the stream names in the string.

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`StreamNames` — Numbering stream names

string

Numbering stream names, specified as a string. To specify multiple stream names, add a space between the stream names in the string.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

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);
```

- “Automatically Number Document Content” on page 13-98

See Also

mlreportgen.dom.AutoNumber | mlreportgen.dom.AutoNumberStream | mlreportgen.dom.CounterReset

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
string

Numbering stream names, specified as a string. To specify multiple stream names, add a space between the stream names in the string.

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`StreamName` — Numbering stream names

string

Numbering stream name, specified as a string. To specify multiple stream names, add a space between the stream names in the string.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

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);
```

- “Automatically Number Document Content” on page 13-98

See Also

mlreportgen.dom.AutoNumber | mlreportgen.dom.AutoNumberStream | mlreportgen.dom.CounterInc

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Properties` — CSS properties

array of `mlreportgen.dom.CSSProperty` objects

CSS properties, specified as an array of `mlreportgen.dom.CSSProperty` objects.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

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.CSSProperty | mlreportgen.dom.FOProperties

More About

- “Report Formatting Approaches” on page 13-21

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/.

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

string

CSS property name, specified as a string.

`Value` — Property value

string

Property value for the corresponding property name, specified as a string.

Output Arguments

`prop` — CSS format object

`mlreportgen.dom.CSSproperty` object

CSS format object, returned as an `mlreportgen.dom.CSSProperty` object.

Properties

`Children` — Children of this object

cell array of objects

This read-only property lists child elements of this object.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Name` — CSS property name

string

CSS property name, specified as a string.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Property value

string

Property value for corresponding property name, specified as a string.

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

More About

- “Report Formatting Approaches” on page 13-21

External Websites

- 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

string

Attribute name, specified as a string.

`value` — Attribute value

string

Attribute value, specified as a string.

Output Arguments

`customAttributeObj` — Custom attribute

`mlreportgen.dom.CustomAttribute` object

Custom attribute, represented by an `mlreportgen.dom.CustomAttribute` object.

Properties

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Name` — Attribute name

string

Attribute name, specified as a string.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Value of this attribute

string

Value of this attribute, specified as a string.

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
string

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

The output format must support the custom attributes of this document element.

`Id` — ID for document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Name` — Element name
string

Element name, specified as a string.

`Style` — Format specification
array of format objects

This property is ignored.

`StyleName` — Name of custom element style
string

This property is ignored.

`Tag` — Tag for document element
string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Methods

Method	Purpose
append	Append a custom element to the document element

Method	Purpose
`clone` Use `CustomElement.clone` similar to how you use `Paragraph.clone`.	Copy custom element.

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('test','html');
```

See Also

mlreportgen.dom.CustomAttribute | mlreportgen.dom.CustomText

mlreportgen.dom.CustomText class

Package: mlreportgen.dom

Plain text appended to custom element

Description

Plain text string 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 string.

Input Arguments

`text` — Text string to append to custom element
string

Text specified as a string.

Output Arguments

`customTextObj` — Text string to append to custom element
`mlreportgen.dom.CustomText` object

Text to append to a custom element, represented by an `mlreportgen.dom.CustomText` object.

Properties

`Id` — ID for document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Text to add

string

Text to add to a custom element, specified as a string.

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.CustomAttribute | mlreportgen.dom.CustomElement

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
string

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 document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Source` — Source object message originates from

a DOM object

Source DOM object from which the message originates.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Text` — Text of the message

string

Message text, specified as a string.

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 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);
```

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);
```

Run report.

```
close(d);
rptview('test','html');
```

Delete the listener to avoid duplicate reporting of message objects during a MATLAB session.

```
delete(l);
```

- “Display Report Generation Messages” on page 13-119

See Also

mlreportgen.dom.MessageDispatcher.dispatch

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

string

Display option, specified as a string. 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 `<span>` element). (Default)
`block`	Block element (similar to an HTML `<span>` element).
`flex`	Block-level flex container.
`initial`	Uses the default value of `inline`.

Value	Display of Text or Paragraph Object
`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 `<li>` 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 **File** > **Options** > **Display** > **Hidden text**, the text displays in the report.
`run-in`	As `block` or `inline`, depending on the context. For example, if the object is inside a block, the object displays as a block.
`table`	Similar to an HTML `<table>` element.
`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 http://www.w3schools.com/cssref/pr_class_display.asp.

Properties

`Id` — ID for `Display` object

string

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

string

Tag for `Display` object, specified as a string.

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

string

Display option, specified as a string. 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

More About

- “Report Formatting Approaches” on page 13-21

External Websites

- http://www.w3schools.com/cssref/pr_class_display.asp
- www.w3schools.com/tags
- www.w3schools.com/cssref

Introduced in R2015b

mlreportgen.dom.Document class

Package: mlreportgen.dom

Document container

Description

Create `mlreportgen.dom.Document` object that defines:

- The type of output: HTML, Microsoft Word, or PDF.
- Where and how to store the output.
- The template to use to format the document.

Construction

`documentObj = Document()` creates an HTML document named `Untitled.htmx` in the current directory, using the default HTML template.

Append content and use a corresponding `close` command to generate the document.

`documentObj = Document(outputPath)` creates an HTML document at the specified location.

`documentObj = Document(outputPath,type)` creates a document of the specified type (for example, Word), using the default template for that type.

`documentObj = Document(outputPath,type,templatePath)` creates a document, using the specified document type and Word, PDF, or HTML template corresponding to the `type` setting.

Input Arguments

`outputPath` — Path for the output file generated by the document

string

Full path of output file or folder for this document. If you do not specify a file extension, the extension is based on the document type (for example, `.docx` for Microsoft Word). You can set this property only before opening the document.

How you specify the path depends on the value of the `PackageType` property.

- `zipped` — Name of zip file.
- `unzipped` — Folder for the output files.
- `both` — Name of zip file.

Data Types: `char`

`type` — Type of output

`'html'` (default) | `'docx'` | `'pdf'` | `'html-file'`

Type of output, specified as `'html'`, `'docx'`, `'pdf'`, or `'html-file'`.

- `'html'`— HTML output as a zipped or unzipped folder containing the HTML document text, image, style sheet, and JavaScript files.
- `'docx'`— Word output.
- `'pdf'`— PDF based on a PDF template.
- `'html-file'`— HTML output consisting of a single file that contains the text, style sheets, JavaScript, and images for the report

If you specify a template using the `templatePath` input argument, the template must be consistent with the `type` argument.

`templatePath` — Path of a custom template

`[ ]` (default) | string

Full path of a custom template file or folder, specified as a string. If you omit a file extension, the template type is based on the document type, for example, `.docx` for Word.

Data Types: `char`

Output Arguments

`documentObj` — Report definition document

`mlreportgen.dom.Document` object

Report definition document, represented by an `mlreportgen.dom.Document` object.

Properties

`Children` — Children of this object
cell array of objects

This read-only property lists child elements of this object.

`CurrentHoleId` — ID of current hole in document
string

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 for a report from this document. If this property is `false` and a writable file of the same name exists, attempting to close (i.e., write) this document 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

string

Custom content for HTML header, specified as a string. 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's template. Set this property only before opening the document.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`OutputPath` — Path of output file or folder for this document

string

Path of this document's output file, specified as a string. If you do not specify the file extension, the DOM adds an extension based on the output type of the document. 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 from this document

`'zipped'` (default) | `'unzipped'` | `'both'`

Packaging for output files, specified as one of these values:

- `zipped` (default)
- `unzipped`
- `both`

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, `pdftx` for PDF, or `htmx` 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`.

The `unzipped` option applies only to Word (`docx`) and multifile HTML (`htmx`) output. 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`, for Word and multifile HTML output. generating the report produces zipped and unzipped output.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`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 when the document is generated (default) | custom tag

String 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.

Data Types: `char`

`TemplatePath` — Path of the template to use

string

The full path to the template to use, specified as a string.

`TitleBarText` — Title for HTML browser title bar

string

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' | 'pdf' | 'html-file'

Type of output, specified as 'html', 'docx', 'pdf', or 'html-file'.

- 'html' — HTML output as a zipped or unzipped folder containing the HTML document text, image, style sheet, and JavaScript files.
- 'docx' — Word output.
- 'pdf' — PDF output.
- 'html-file' — HTML output consisting of a single 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 Word template (.dotx) for docx output, an HTML template package (.htmtx) for HTML output, a PDF template package (.pdftx) for PDF output, and a single-file HTML template (.html) for html-file output.

Methods

Method	Purpose
addHTML	Append HTML string to document
addHTMLFile	Append HTML file contents to document
append	Append document element to the document.
close	Close this document.
createAutoNumberStream	Create automatically generated numbering stream.
createTemplate	Create document template.
fill	Fill document hole.
getAutoNumberStream	Get the automated numbering stream.
getCoreProperties	Get core properties of document.
getMainPartPath	Get relative path of main part of output document.

Method	Purpose
getOPCMainPart	Get full path of main part of output document.
moveToNextHole	Move to next template hole.
open	Open this document.
package	Append file to OPC package of document.
setCoreProperties	Set core properties of document element.

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.OutputPath);
```

Create an HTML Document as a Single File

Create an HTML document as a single HTML file that includes the images of the document. The example assumes that there is a `MyImage.jpg` image 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 the document and view it.

```
import mlreportgen.dom.*;
d = Document('mydoc','html-file','myHTMLTemplate');
open(d);

append(d,'Hello world');
append(d,Image('C:/images/LocalSystem/MyImage.jpg'));

close(d);
```

```
rptview(d.OutputPath);
```

See Also

mlreportgen.dom.DocumentPart

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) | string

Full path of this part's template file or folder, specified as a string. 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
string

Document part template name, specified as a string. Specify where the part is stored using the `templatePath` or `templateSrc` argument.

`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 a `mlreportgen.dom.DocumentPart` object for a document part.

Output Arguments

`documentPartObj` — Document part
`mlreportgen.dom.DocumentPart` object

Document part, returned as an `mlreportgen.dom.DocumentPart` object.

Properties

`Children` — Children of this object
cell array of objects

This read-only property lists child elements of this object.

`CurrentHoleId` — ID of current hole in document

string

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 [].

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`TemplateName` — Name of this part's template

string

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

string

Path of this part's template or of a template whose template library contains this part's template, specified as a string.

`Type` — Output format

`'html' | 'docx' | 'pdf'`

Format for the output, specified as `'html'`, `'docx'`, or `'pdf'`. 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 `DocumentPart.addHTML` in a similar way to how you use `Document.addHTML`.	Append HTML string to document
addHTMLFile Use `DocumentPart.addHTMLFile` in a similar way to how you use `Document.addHTMLFile`.	Append HTML file contents to document

Method	Purpose
append	Append document element to the document part.
close	Close this document part.
createTemplate	Create document part template.
fill	Fill document hole.
getCoreProperties	Get core properties of document part.
getOPCMainPart	Get full path of main part of output document.
moveToNextHole	Move to next template hole.
open	Open this document part.
setCoreProperties	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');
```

- “Use Subforms in a Report” on page 13-35

See Also

mlreportgen.dom.Document

More About

- “Form-Based Reporting” on page 13-32

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

`docxPageFooterObj = DOCXPageFooter()` creates an empty page footer based on the default Word template.

`docxPageFooterObj = DOCXPageFooter(pageType,templatePath)` creates an empty page footer for the specified type of page based on the specified template.

`docxPageFooterObj = DOCXPageFooter(pageType,templatePath, docPartTemplateName)` creates an empty page footer for the specified type of page, based on the specified document part template in the specified template.

`docxPageFooterObj = DOCXPageFooter(pageType,templateSrc, docPartTemplateName)` creates an empty 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` — If there are no page footers defined with `pageType` set to `first`, the footer appears on the odd pages of a section and on the first page.

- `first` — Footer appears only on the first page of a section.
- `even` — Footer appears on the even pages of a section.

For example, to make different footers appear on the first page and on even pages, define two footers. Set `pageType` to `first` for one and to `even` for the other.

`templatePath` — Full path of footer template
string

Full path of footer template, specified as a string.

`docPartTemplateName` — Document part template name
string

Name of this part's template if it is stored in a template specified by the `templatePath` or `templateSrc` argument, specified as a string.

`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 a `mlreportgen.dom.DocumentPart` object for a document part.

Output Arguments

`docxPageFooterObj` — 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 this object
cell array of objects

This read-only property lists child elements of this object.

`CurrentPageLayout` — Ignored by page footers

`CurrentHoleId` — ID of current hole in document
string

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 document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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`— If there are no page footers defined with `pageType` set to `first`, footer appears on odd pages in a section and the first page.
- `first`— Appears only on the first page of a section.
- `even`— Appears on even pages in a section.

To have a footer appear on the first page and on even pages, define two footers, one with `pageType` set to `first` and the other with `pageType` set to `even`.

`Parent` — Parent of document element
DOM object

This read-only property lists the parent of this document element.

`Tag` — Tag for document element
string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`TemplatePath` — Path to template used for footer

string

Full path to the template to use for this footer, specified as a string.

Methods

Use `DocumentPageFooter` methods like you use the corresponding `Document` methods.

Method	Purpose
append	Append one of these DOM objects to the footer: • `CustomElement` • `FormalTable` • `Group` • `ExternalLink` • `Image` • `InternalLink` • `OrderedList` • `Paragraph` • `RawText` • `Table` • `Text` • `UnorderedList`
close	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);
```

- “Create Page Footers and Headers” on page 13-152

See Also

mlreportgen.dom.DOCXPageHeader | mlreportgen.dom.DOCXPageLayout | mlreportgen.dom.PDFPageFooter | mlreportgen.dom.PDFPageHeader | mlreportgen.dom.PDFPageLayout

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

`docxPageHeaderObj = DOCXPageHeader()` creates an empty page header based on the default Word template.

`docxPageHeaderObj = DOCXPageHeader(pageType,templatePath)` creates an empty page header for the specified type of page based on the specified template.

`docxPageHeaderObj = DOCXPageHeader(pageType,templatePath, docPartTemplateName)` creates an empty page header for the specified type of page, based on the specified document part template in the specified template.

`docxPageHeaderObj = DOCXPageHeader(pageType,templateSrc, docPartTemplateName)` creates an empty 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 appears on odd pages in a section and the first page, if there are no page headers defined with `pageType` set to `first`.
- `first`— Header appears only on the first page of a section.
- `even`— Header appears on even pages in a section.

For example, to have a header appear on the first page and on even pages, define two headers, one with `pageType` set to `first` and the other with `pageType` set to `even`.

`templatePath` — Full path of header template
string

Full path of header template, specified as a string.

`docPartTemplateName` — Document part template name
string

Name of this part's template if it is stored in a template specified by the `templatePath` or `templateSrc` argument, specified as a string.

`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 a `mlreportgen.dom.DocumentPart` object for a document part.

Output Arguments

`docxPageHeaderObj` — Page header for Word document
`mlreportgen.dom.DOCXPageHeader` object

Page header for a Word document, returned as an `mlreportgen.dom.DOCXPageHeader` object.

Properties

`Children` — Children of this object
cell array of objects

This read-only property lists child elements of this object.

`CurrentHoleId` — ID of current hole in document
string

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 [].

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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 appears on odd pages in a section and the first page, if there are no page headers defined with `pageType` set to `first`.
- `first`— Header appears only on the first page of a section.
- `even`— Header appears on even pages in a section.

For example, to have a header appear on the first page and on even pages, define two headers, one with `pageType` set to `first` and the other with `pageType` set to `even`.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`TemplatePath` — The path to template used for header

string

Full path to the template to use for this header, specified as a string.

Methods

Use `DOCXPageHeader` methods like you use the corresponding `Document` methods.

Method	Purpose
append	Append one of these DOM objects to the header: • `CustomElement` • `DOCXPageLayout` • `FormalTable` • `Group` • `ExternalLink` • `Image` • `InternalLink` • `OrderedList` • `Paragraph` • `RawText` • `Table`

Method	Purpose
	• `Text` • `UnorderedList`
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.PDFPageFooter | mlreportgen.dom.PDFPageHeader | mlreportgen.dom.PDFPageLayout

Related Examples

- “Create Page Footers and Headers” on page 13-152

mlreportgen.dom.DOCXPageLayout class

Package: mlreportgen.dom

Page format and layout for Microsoft Word document section

Description

Use an `mlreportgen.dom.DOCXPageLayout` object to define the page format, headers, and footers of a Word document section.

Construction

`PageLayoutObj = DOCXPageLayout()` creates an empty document page layout object.

Output Arguments

`PageLayoutObj` — Page layout object
`mlreportgen.dom.DOCXPageLayout` object

Page format and layout for Word document section, returned as an `mlreportgen.dom.DOCXPageLayout` object.

Properties

`Children` — Children of this object
cell array of objects

This read-only property lists child elements of this object.

`CustomAttributes` — Custom attributes of document element
array of `mlreportgen.dom.CustomAttribute` objects

The output format must support the custom attributes of this document element.

`FirstPageNumber` — Number of first page in section
integer

Number of the first page in a section, specified as an integer.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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 `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

`PageMargins` — Margin sizes for this layout

`mlreportgen.dom.PageMargins` object

Margin sizes for this layout, specified as an `mlreportgen.dom.PageMargins` object.

`PageNumberFormat` — Type of page numbering to use

`'a' | 'A' | 'i' | 'I' | 'n' | 'N'`

Type of page numbering to use, specified as one of these values:

- `'a'` — Lowercase alphabetic.
- `'A'` — Uppercase alphabetic.
- `'i'` — Lowercase roman numerals.
- `'I'` — Uppercase roman numerals.
- `'n'` or `'N'` — Numeric page numbers.

`PageSize` — Size and orientation of pages in this layout

`mlreportgen.dom.PageSize` object

Size and orientation of pages in this layout, specified as an `mlreportgen.dom.PageSize` object.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`RawFormats` — XML markup for unsupported layout formats

cell array

XML markup for unsupported section formats, specified as a cell array of strings. For information about XML markup for Word formats, see http://officeopenxml.com/anatomyofOOXML.php.

`SectionBreak` — Section break options

`'Next Page'` | `'Same Page'` | `'Odd Page'` | `'Even Page'`

Option to create a section break for this layout, specified as one of these values:

- `'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.

`Style` — Formats to apply to layout

array of format objects

Formats to apply to this layout, specified as an array of format objects. The formats you specify using this property override the same formats defined by the style applied with the `StyleName` property. Formats that do not apply to a page layout are ignored.

`StyleName` — Ignored by page layouts

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Examples

Change Page Margins of a Document Section

Create a Word report using the default template. Open the document and assign the document's `CurrentPageLayout` property to a variable. Change the left and right margins for this layout.

```
import mlreportgen.dom.*;
d = Document('mydoc','docx');

open(d);
s = d.CurrentPageLayout;
s.PageMargins.Left = '2in';
s.PageMargins.Right = '2in';
p = Paragraph('Hello World');
append(d,p);

close(d);
rptview('mydoc.docx');
```

See Also

mlreportgen.dom.DocumentPart | mlreportgen.dom.DOCXPageFooter | mlreportgen.dom.DOCXPageHeader | mlreportgen.dom.DOCXSubDoc | mlreportgen.dom.PageMargins | mlreportgen.dom.PageRawFormat | mlreportgen.dom.PageSize

More About

- "Report Formatting Approaches" on page 13-21

Introduced in R2016a

mlreportgen.dom.DOCXSubDoc class

Package: mlreportgen.dom

Reference to external Microsoft Word document

Description

Reference to external Microsoft Word document.

Construction

`docxSubDocObj = DOCXSubDoc()` creates an empty document reference.

`docxSubDocObj = DOCXSubDoc(path)` creates a reference to a Word document at the specified path. Appending this reference to a Word document (the master document) inserts a link to the subdocument at the point at which the reference is appended.

Opening a master document in Word causes the link to the subdocument to be displayed, instead of its content. To replace the link with the content, select **Expand Subdocuments** from the **Outlining** tab of the **View** tab on the Word tool strip. The `rptview` command expands subdocuments automatically when it opens a Word document.

Input Arguments

`path` — Path of document targeted by this reference

string

Path of document targeted by this reference, specified as a string.

Output Arguments

`docxSubDocObj` — Reference to external Word document

`mlreportgen.dom.DOCXSubDoc` object

Path of Word document targeted by this reference, represented by an `mlreportgen.dom.DOCXSubDoc` object.

Properties

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Target` — Path of document targeted by this reference

string

Path of document targeted by this reference, specified as a string. Use ASCII characters. Use the following format for specifying a full path involving a mapped drive.

```
'file:///C:/UserPath/FileName.docx'
```

Methods

Method	Purpose
clone Use `DOCXSubDoc.clone` in a similar way to how you use `Paragraph.clone`.	Clone this Word document reference.

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);
```

See Also

mlreportgen.dom.DocumentPart | mlreportgen.dom.DOCXSection

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
string

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 document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Source` — Source object from which the message originates

a DOM object

Source DOM object from which the message originates.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Text` — Text of message

string

Message text, specified as a string.

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);
dispatch(dispatcher,ErrorMessage('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);
rptview('test','html');
```

Delete the listener to avoid duplicate reporting of message objects during a MATLAB session.

```
delete(l);
```

- “Display Report Generation Messages” on page 13-119

See Also

mlreportgen.dom.MessageDispatcher.dispatch

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

string | `mlreportgen.dom.LinkTarget` object

The link target of the external link, specified as either a string (for a URL) or as an `mlreportgen.dom.LinkTarget` object.

`linkText` — Link text

string

The text to use for the link text.

`linkTextStyleName` — Name of style for link text

string

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.CustomAttribute` objects

The output format must support the custom attributes of this document element.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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

string

Name of link style defined in the template, specified as a string. 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 document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Target` — Target URL of link

string

This read-only property displays the URL of the link target of this hyperlink.

Methods

Method	Purpose
append	Append text or a `Text`, `Image`, or `CustomElement` object.
clone Use `ExternalLink.clone` in a similar way to how you use `Paragraph.clone`.	Copy the external link

Examples

Add an External Link

```
import mlreportgen.dom.*
d = Document('mydoc');

append(d,ExternalLink('http://www.mathworks.com/','MathWorks'));

close(d);
rptview('mydoc','html');
```

- “Create Links” on page 13-83

See Also

mlreportgen.dom.InternalLink | mlreportgen.dom.LinkTarget

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

string

String specifying the width of indentation of first line of paragraph. String must have the format `valueUnits` where `Units` is an abbreviation for the units in which the size is expressed. Valid abbreviations are:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas

- `pt` — points
- `px` — pixels

`style` — Type of indentation
string

String specifying the style of indentation of the first line of the paragraph.

- `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 document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Style` — Type of indentation
string

String specifying the style of indentation of the first line of the paragraph.

- `normal` (default) — indent relative to the page margin
- `hanging` — indent relative to the subsequent lines

`Tag` — Tag for document element
string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Width` — Amount of indentation

string

String specifying the width of indentation of first line of paragraph. String must have the format `valueUnits` where `Units` is an abbreviation for the units in which the size is expressed. Valid abbreviations are:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

See Also

mlreportgen.dom.Paragraph

More About

- “Report Formatting Approaches” on page 13-21

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` — Control text flow or table column ordering

string

String specifying the direction for text to flow or for table columns to appear.

- `'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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Text flow direction

string

String specifying the direction for text to flow or the order of table columns.

- `'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,doctype);
```

- “Create and Format Paragraphs” on page 14-84

See Also

mlreportgen.dom.Paragraph | mlreportgen.dom.TextOrientation

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

string

Font family, specified as a string.

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 strings 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

string

For Word documents only. String specifying a font family to substitute in a locale that requires a complex script (such as Arabic) to render text.

`EastAsiaFamilyName` — Font family for East Asian locales

string

For Word documents only. String specifying a font family to substitute in an East Asian locale, such as China, Japan, or Korea.

`FamilyName` — Font family to use if possible

string

String specifying a font family to be used for document text.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

See Also

mlreportgen.dom.CharEntity | mlreportgen.dom.FontSize | mlreportgen.dom.Paragraph | mlreportgen.dom.Text

More About

- “Report Formatting Approaches” on page 13-21

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) | string

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the size is expressed. The following abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

Output Arguments

`fontSizeObj` — Font size
`mlreportgen.dom.FontSize` object

Font size, represented by an `mlreportgen.dom.FontSize` object.

Properties

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Font size

`'12pt'` (default) | string

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the size is expressed. The following abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

See Also

mlreportgen.dom.FontFamily | mlreportgen.dom.Paragraph | mlreportgen.dom.Text

More About

- "Report Formatting Approaches" on page 13-21

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.

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 document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Parent` — Parent of document element
DOM object

This read-only property lists the parent of this document element.

`Properties` — FO properties
array of `mlreportgen.dom.FOProperty` objects

FO properties, specified as an array of `mlreportgen.dom.FOProperty` objects.

`Tag` — Tag for document element
string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

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.Page | mlreportgen.dom.Paragraph

More About

- “Report Formatting Approaches” on page 13-21

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.

Construction

`prop = FOProperty(Name,Value)` creates an FO format property having the specified name and value.

Input Arguments

`Name` — FO property name

string

FO property name, specified as a string.

`Value` — Property value

string

Property value for the corresponding property name, specified as a string.

Output Arguments

`prop` — FO format object

`mlreportgen.dom.FOproperty` object

FO format object, returned as an `mlreportgen.dom.FOProperty` object.

Properties

`Children` — Children of this object

cell array of objects

This read-only property lists child elements of this object.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Name` — FO property name

string

FO property name, specified as a string.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Property value

string

Property value for the corresponding property name, specified as a string.

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.CSSProperties | mlreportgen.dom.FOProperties | mlreportgen.dom.PageBreakBefore

More About

- “Report Formatting Approaches” on page 13-21

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 or table footer, or both. The table header, body, and footer are `mlreportgen.dom.TableHeader`, `mlreportgen.dom.TableBody`, and `mlreportgen.dom.TableFooter` objects, respectively.

Construction

`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 whose body content you specify as a two-dimensional array. The constructor converts basic MATLAB types to corresponding DOM objects. For example, the constructor converts strings 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 cell array

The cell array may contain:

- Character array (strings)
- 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

string

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 strings

The cell array may contain:

- Character array (strings)
- 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 strings

The cell array may contain:

- Character array (strings)
- 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

Output Arguments

`formalTableObj` — Formal table

`mlreportgen.dom.FormalTable` object

Formal table, represented by an `mlreportgen.dom.FormalTable` object.

Properties

`BackgroundColor` — Background color

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`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

string

Specify one of the following as a string.

Border String	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

Border String	Description	Supported 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 See discussion below this table.	HTML and PDF
`inset`	3-D effect line	All output types
`none`	No line See discussion below this table.	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

Border String	Description	Supported Output Types
`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)

string

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

string

You can specify:

- Name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.

- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`BorderWidth` — Table border width

string

String specifying the width of the border. The string 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`Children` — Children of this object

cell array of objects

This read-only property lists child elements of this object.

`ColSep` — Style of line separating columns

string

The style the line separating the columns of a table or table section (header, body, footer), as specified by a `mlreportgen.dom.ColSep` object.

See the description of the `Border` property for a description of the possible values.

`ColSepColor` — Color of line separating columns

string

You can specify:

- Name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`ColSepWidth` — Width of line separating table columns

string

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the width is expressed. Use one of these abbreviations for the units of a width.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

For example, for a three pica wide column separator, set the `ColSepWidth` property to `3pi`.

`ColSpecGroups` — Properties of group of columns in table

array of `mlreportgen.dom.TableColSpecGroups` objects

An array of `mlreportgen.dom.TableColSpecGroups` 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, etc. Specify the number of columns belonging to each group using the `Span` property of the `TableColSpecGroups` 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, etc.

`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

string

String specifying the direction for table columns to flow.

- `'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

string

Possible values are:

- `center`
- `left`
- `right`

`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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`OuterLeftMargin` — Left margin (indentation) of document element

string

String specifying the left indentation. The string 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

- `pi` — picas
- `pt` — points
- `px` — pixels

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`RowSep` — Style of lines separating rows

string

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

string

You can specify:

- Name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`RowSepWidth` — Width of row separator

string

String specifying the width of the row separator. The string 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`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 stylesheet style specified by the `StyleName` property.

`StyleName` — Style in document or document part stylesheet

string

Name of a style specified in the stylesheet 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

string

The inner margin is the margin between table cell content and the cell borders.

The string 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
- `pi` — picas

- `pt` — points
- `px` — pixels

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Width` — Table width

string

String representing 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`, where `Units` is an abbreviation for the units in which the width is expressed.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

Methods

Method	Purpose
`append` Use `FormalTable.append` similar to how you use `TableRow.append`.	Append a row of table entries to table.

Method	Purpose
`clone` Use `FormalTable.clone` the same way you use `Paragraph.clone`.	Copy the table.

See Also

mlreportgen.dom.ResizeToFitContents | mlreportgen.dom.Table | mlreportgen.dom.TableBody | mlreportgen.dom.TableColSpec | mlreportgen.dom.TableEntry | mlreportgen.dom.TableFooter | mlreportgen.dom.TableHeader | mlreportgen.dom.TableRow

Related Examples

- “Create and Format Tables” on page 13-69

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Children` — Children of this object

cell array of objects

This read-only property lists child elements of this object.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Methods

Method	Purpose
append	Append a DOM object to the group

See Also

mlreportgen.dom.Container

Related Examples

- "Add Content as a Group" on page 13-14
- "Create Object Containers" on page 13-30

mlreportgen.dom.HAlign class

Package: mlreportgen.dom

Specify horizontal alignment of document object

Description

Specifies horizontal alignment of a document object.

Construction

`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

string

String that specifies the horizontal alignment of a document object.

- `'center'`
- `'left'`
- `'right'`

Output Arguments

`horizontalAlignObj` — Horizontal alignment

`mlreportgen.dom.HAlign` object

Horizontal alignment, represented by an `mlreportgen.dom.HAlign` object.

Properties

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Horizontal alignment

string

- `'center'`
- `'left'`
- `'right'`

See Also

mlreportgen.dom.VAlign

More About

- “Report Formatting Approaches” on page 13-21

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` — Text string

string

Text to use for the heading.

`styleName` — Style for text

string

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

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.

- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`Bold` — Option to use bold for text

[ ] (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 text 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

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`CustomAttributes` — Custom attributes of document element

array of `mlreportgen.dom.CustomAttribute` objects

The output format must support the custom attributes of this document element.

`FirstLineIndent` — Indentation amount for first line of paragraph

string

Amount by which to indent the first line of this paragraph relative to succeeding lines.

To create a hanging indent, in which all the lines are indented except for the first line, use a negative number.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the indentation is expressed. Use one of these abbreviations for the units for indentation.

- no abbreviation — pixels

- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`FontFamilyName` — Name of font family for text

string

The name of a font family.

To specify substitutions for this font, do not set this property. Instead create and add a `mlreportgen.dom.FontFamily` object to the `Style` property of this document element.

`FontSize` — Font size for text

string

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the font size is expressed. Use one of these abbreviations for the units for the font size.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`HAlign` — Horizontal alignment of this paragraph

string

Horizontal alignment for a paragraph, relative to page margins or table cell borders, specified as a string.

String	Description	Supported Output Types
`center`	Center the paragraph	All output types

String	Description	Supported Output Types
`distribute`	Distribute all characters equally	Word
`justify`	Align left side of paragraph on left side of page, and right side of paragraph on the right side of the page	HTML and PDF
`KashidaHigh`	Use widest Kashida length	Word
`KashidaLow`	Use lowest Kashida length	Word
`KashidaMedium`	Use medium Kashida length	Word
`left`	Align paragraph left	All output types
`right`	Align paragraph right	All output types
`ThaiDistribute`	Thai language justification	Word

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Italic` — Option to use italics for text

[ ] (default) | logical value

To use italics for text, 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 text 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`

`OuterLeftMargin` — Left margin (indentation) of document element

string

String specifying the left indentation. The string 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`OutlineLevel` — Outline level of this paragraph

[ ] (default) | numeric value

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 table of contents. For example, to make a paragraph appear as a `Heading 1` (Word) or `h1` (HTML), set `OutlineLevel` to `1`.

Data Types: `int32`

`Strike` — Text strikethrough

string

The default for this property is []. You can set it to one of these values:

- `none` — Do not use strikethrough.
- `single` — Use a single line for strikethrough.
- `double` — Use a double line for strikethrough for Word documents.

Setting the `Strike` property adds a corresponding `mlreportGen.dom.Strike` format object to the `Style` property for this document element. Removing the `Strike` property setting 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

string

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 document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Underline` — Type of underline, if any, for text

[ ] (default) | string

You can specify one of the following types of underlines.

Border String	Description	Supported Output Types
`dash`	Dashed underline	Word
`dashedHeavy`	Line with heavy dashes	Word
`dashLong`	Line with long dashes	Word
`dashLongHeavy`	Line with heavy long dashes	Word
`dashDotDotHeavy`	Line with heavy dashes with two dots between the dashes	Word
`dashDotHeavy`	Heavy dash-dot line	Word
`dotted`	Dotted line	Word
`dottedHeavy`	Thick dotted line	Word
`dotDash`	Dot-dash line	Word
`dotDotDash`	Dot-dot-dash line	Word
`dashDotHeavy`	Heavy dot-dash line	Word
`double`	Double line	Word
`none`	Do not use underlining	All output types
`single`	Single line	All output types
`thick`	Thick line	Word

Border String	Description	Supported Output Types
`wave`	Wavy line	Word
`waveyDouble`	Double wavy line	Word
`waveyHeavy`	Heavy wavy	Word
`words`	Underline non-space characters only	Word

If this property is empty and `StyleName` property of this document element specifies a style sheet style, the type of underline is determined by that style.

To specify the color as well as the type of the underline, do not set the `Underline` property. Instead, set the `Style` property of this document element to include an `mlreportgen.dom.Underline` format object that specifies the desired underline type and color.

Setting the `Underline` property adds a corresponding `mlreportgen.dom.Underline` format object to the `Style` property for this document element. Removing the `Underline` property setting removes the object.

`WhiteSpace` — White space and line breaks in text

[ ] (default) | string

To specify how to handle white space, use one of these values.

String	Description	Supported Output Types
`normal` (default)	For HTML and PDF, removes spaces at beginning and end of text string. Multiple spaces within string collapse to single space. For Word, removes spaces at beginning and end of text string.	All output types
`nowrap`	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML

String	Description	Supported Output Types
`preserve`	Preserves spaces and line feeds. Acts like the `<pre>` tag in HTML.	All output types
`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.	HTML and PDF
`pre-wrap`	Preserves white space. Text wraps when necessary and on line breaks	HTML and PDF

Setting the `WhiteSpace` property adds a corresponding `WhiteSpace` format object to `Style` property. Removing the `WhiteSpace` property setting removes the `WhiteSpace` object.

Methods

Method	Purpose
append Use `Heading.append` similar to how you use `Paragraph.append`.	Append content to heading.
clone Use `Heading.clone` similar to how you use `Paragraph.clone`.	Copy heading.

See Also

mlreportgen.dom.Heading1 | mlreportgen.dom.Paragraph

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

string

Heading text, specified as a string.

`styleName` — Style for the heading

string

The name of a style, specified as a string. 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

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`Bold` — Option to use bold for text

[ ] (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 text 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`

`Children` — Children of this object

cell array of objects

This read-only property lists child elements of this object.

`Color` — Text color

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`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

string

Amount by which to indent the first line of this paragraph relative to succeeding lines.

To create a hanging indent, in which all the lines are indented except for the first line, use a negative number.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the indentation is expressed. Use one of these abbreviations for the units for indentation.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`FontFamilyName` — Name of font family for text

string

The name of a font family.

To specify substitutions for this font, do not set this property. Instead create and add a `mlreportgen.dom.FontFamily` object to the `Style` property of this document element.

Setting the `FontFamilyName` property adds a corresponding `mlreportGen.dom.FontFamily` format object to the `Style` property for this document element. Removing the `FontFamilyName` property setting removes the object.

`FontSize` — Font size for text

string

If you need to specify substitutions for this font, do not set this property. Instead create and add a `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.

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the font size is expressed. Use one of these abbreviations for the units for the font size.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Italic` — Option to use italics for text

[ ] (default) | logical value

To use italics for text, 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 text 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`

`OuterLeftMargin` — Left indentation for paragraph

string

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.

To indent a paragraph from both the left and right margin of a page, do not set this property. Instead, add to the `Style` property of this paragraph a `mlreportgen.dom.OuterMargin` object specifying the left and right indentations.

Setting the `OuterLeftMargin` property adds a corresponding `mlreportGen.dom.OuterMargin` format object to the `Style` property for this document element. Removing the `OuterLeftMargin` property setting removes the object.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the indentation is expressed. Use one of these abbreviations for the units for indentation.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`OutlineLevel` — Outline level of this paragraph

[ ] (default) | numeric value

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 table of contents. For example, to make a paragraph appear as a `Heading 1` (Word) or `h1` (HTML), set `OutlineLevel` to `1`.

Data Types: `int32`

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Strike` — Text strikethrough

string

The default for this property is []. You can set it to one of these values:

- `none` — Do not use strikethrough.
- `single` — Use a single line for strikethrough.
- `double` — Use a double line for strikethrough for Word documents.

Setting the `Strike` property adds a corresponding `mlreportGen.dom.Strike` format object to the `Style` property for this document element. Removing the `Strike` property setting 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 stylesheet

string

Name of the style to apply from the stylesheet, specified as a string.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Underline` — Type of underline, if any, for text

[ ] (default) | string

You can specify one of the following types of underlines.

Border String	Description	Supported Output Types
`dash`	Dashed underline	Word
`dashedHeavy`	Line with heavy dashes	Word
`dashLong`	Line with long dashes	Word
`dashLongHeavy`	Line with heavy long dashes	Word
`dashDotDotHeavy`	Line with heavy dashes with two dots between the dashes	Word
`dashDotHeavy`	Heavy dash-dot line	Word
`dotted`	Dotted line	Word
`dottedHeavy`	Thick dotted line	Word
`dotDash`	Dot-dash line	Word
`dotDotDash`	Dot-dot-dash line	Word
`dashDotHeavy`	Heavy dot-dash line	Word
`double`	Double line	Word
`none`	Do not use underlining	All output types
`single`	Single line	All output types
`thick`	Thick line	Word
`wave`	Wavy line	Word
`waveyDouble`	Double wavy line	Word
`waveyHeavy`	Heavy wavy	Word
`words`	Underline non-space characters only	Word

If this property is empty and `StyleName` property of this document element specifies a style sheet style, the type of underline is determined by that style.

To specify the color as well as the type of the underline, do not set the `Underline` property. Instead, set the `Style` property of this document element to include an `mlreportgen.dom.Underline` format object that specifies the desired underline type and color.

Setting the `Underline` property adds a corresponding `mlreportgen.dom.Underline` format object to the `Style` property for this document element. Removing the `Underline` property setting removes the object.

`WhiteSpace` — White space and line breaks in text

[ ] (default) | string

To specify how to handle white space, use one of these values.

String	Description	Supported Output Types
`normal` (default)	For HTML and PDF, removes spaces at beginning and end of text string. Multiple spaces within string collapse to single space. For Word, removes spaces at beginning and end of text string.	All output types
`nowrap`	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
`preserve`	Preserves spaces and line feeds. Acts like the `<pre>` tag in HTML.	All output types
`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.	HTML and PDF

String	Description	Supported Output Types
`pre-wrap`	Preserves white space. Text wraps when necessary and on line breaks	HTML and PDF

Setting the `WhiteSpace` property adds a corresponding `WhiteSpace` format object to `Style` property. Removing the `WhiteSpace` property setting removes the `WhiteSpace` object.

Methods

Method	Purpose
append Use `Heading1.append` similar to how you use `Paragraph.append`.	Append content to heading.
clone Use `Heading1.clone` similar to how you use `Paragraph.clone`.	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.Heading | mlreportgen.dom.Heading2 | mlreportgen.dom.Heading3 | mlreportgen.dom.Heading4 | mlreportgen.dom.Heading5 | mlreportgen.dom.Heading6 | mlreportgen.dom.Paragraph

More About

- “Report Formatting Approaches” on page 13-21

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

string

Heading text, specified as a string.

`styleName` — Style for the heading

string

The name of a style, specified as a string. 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

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`Bold` — Option to use bold for text

[ ] (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 text 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`

`Children` — Children of this object

cell array of objects

This read-only property lists child elements of this object.

`Color` — Text color

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`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

string

Amount by which to indent the first line of this paragraph relative to succeeding lines.

To create a hanging indent, in which all the lines are indented except for the first line, use a negative number.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the indentation is expressed. Use one of these abbreviations for the units for indentation.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`FontFamilyName` — Name of font family for text

string

The name of a font family.

To specify substitutions for this font, do not set this property. Instead create and add a `mlreportgen.dom.FontFamily` object to the `Style` property of this document element.

Setting the `FontFamilyName` property adds a corresponding `mlreportGen.dom.FontFamily` format object to the `Style` property for this document element. Removing the `FontFamilyName` property setting removes the object.

`FontSize` — Font size for text

string

If you need to specify substitutions for this font, do not set this property. Instead create and add a `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.

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the font size is expressed. Use one of these abbreviations for the units for the font size.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Italic` — Option to use italics for text

[ ] (default) | logical value

To use italics for text, 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 text 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`

`OuterLeftMargin` — Left indentation for paragraph

string

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.

To indent a paragraph from both the left and right margin of a page, do not set this property. Instead, add to the `Style` property of this paragraph a `mlreportgen.dom.OuterMargin` object specifying the left and right indentations.

Setting the `OuterLeftMargin` property adds a corresponding `mlreportGen.dom.OuterMargin` format object to the `Style` property for this document element. Removing the `OuterLeftMargin` property setting removes the object.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the indentation is expressed. Use one of these abbreviations for the units for indentation.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`OutlineLevel` — Outline level of this paragraph

[ ] (default) | numeric value

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 table of contents. For example, to make a paragraph appear as a `Heading 1` (Word) or `h1` (HTML), set `OutlineLevel` to `1`.

Data Types: `int32`

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Strike` — Text strikethrough

string

The default for this property is []. You can set it to one of these values:

- `none` — Do not use strikethrough.
- `single` — Use a single line for strikethrough.
- `double` — Use a double line for strikethrough for Word documents.

Setting the `Strike` property adds a corresponding `mlreportGen.dom.Strike` format object to the `Style` property for this document element. Removing the `Strike` property setting 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 stylesheet

string

Name of the style to apply from the stylesheet, specified as a string.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Underline` — Type of underline, if any, for text

[ ] (default) | string

You can specify one of the following types of underlines.

Border String	Description	Supported Output Types
`dash`	Dashed underline	Word
`dashedHeavy`	Line with heavy dashes	Word
`dashLong`	Line with long dashes	Word
`dashLongHeavy`	Line with heavy long dashes	Word
`dashDotDotHeavy`	Line with heavy dashes with two dots between the dashes	Word
`dashDotHeavy`	Heavy dash-dot line	Word
`dotted`	Dotted line	Word
`dottedHeavy`	Thick dotted line	Word
`dotDash`	Dot-dash line	Word
`dotDotDash`	Dot-dot-dash line	Word
`dashDotHeavy`	Heavy dot-dash line	Word
`double`	Double line	Word
`none`	Do not use underlining	All output types
`single`	Single line	All output types
`thick`	Thick line	Word
`wave`	Wavy line	Word
`waveyDouble`	Double wavy line	Word
`waveyHeavy`	Heavy wavy	Word
`words`	Underline non-space characters only	Word

If this property is empty and `StyleName` property of this document element specifies a style sheet style, the type of underline is determined by that style.

To specify the color as well as the type of the underline, do not set the `Underline` property. Instead, set the `Style` property of this document element to include an `mlreportgen.dom.Underline` format object that specifies the desired underline type and color.

Setting the `Underline` property adds a corresponding `mlreportgen.dom.Underline` format object to the `Style` property for this document element. Removing the `Underline` property setting removes the object.

`WhiteSpace` — White space and line breaks in text

[ ] (default) | string

To specify how to handle white space, use one of these values.

String	Description	Supported Output Types
`normal` (default)	For HTML and PDF, removes spaces at beginning and end of text string. Multiple spaces within string collapse to single space. For Word, removes spaces at beginning and end of text string.	All output types
`nowrap`	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
`preserve`	Preserves spaces and line feeds. Acts like the `<pre>` tag in HTML.	All output types
`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.	HTML and PDF

String	Description	Supported Output Types
`pre-wrap`	Preserves white space. Text wraps when necessary and on line breaks	HTML and PDF

Setting the `WhiteSpace` property adds a corresponding `WhiteSpace` format object to `Style` property. Removing the `WhiteSpace` property setting removes the `WhiteSpace` object.

Methods

Method	Purpose
append Use `Heading2.append` similar to how you use `Paragraph.append`.	Append content to heading.
clone Use `Heading2.clone` similar to how you use `Paragraph.clone`.	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.Heading | mlreportgen.dom.Heading1 | mlreportgen.dom.Heading3 | mlreportgen.dom.Heading4 | mlreportgen.dom.Heading5 | mlreportgen.dom.Heading6 | mlreportgen.dom.Paragraph

More About

- “Report Formatting Approaches” on page 13-21

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

string

Heading text, specified as a string.

`styleName` — Style for the heading

string

The name of a style, specified as a string. 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

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`Bold` — Option to use bold for text

[ ] (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 text 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`

`Children` — Children of this object

cell array of objects

This read-only property lists child elements of this object.

`Color` — Text color

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`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

string

Amount by which to indent the first line of this paragraph relative to succeeding lines.

To create a hanging indent, in which all the lines are indented except for the first line, use a negative number.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the indentation is expressed. Use one of these abbreviations for the units for indentation.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`FontFamilyName` — Name of font family for text

string

The name of a font family.

To specify substitutions for this font, do not set this property. Instead create and add a `mlreportgen.dom.FontFamily` object to the `Style` property of this document element.

Setting the `FontFamilyName` property adds a corresponding `mlreportGen.dom.FontFamily` format object to the `Style` property for this document element. Removing the `FontFamilyName` property setting removes the object.

`FontSize` — Font size for text

string

If you need to specify substitutions for this font, do not set this property. Instead create and add a `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.

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the font size is expressed. Use one of these abbreviations for the units for the font size.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Italic` — Option to use italics for text

[ ] (default) | logical value

To use italics for text, 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 text 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`

`OuterLeftMargin` — Left indentation for paragraph

string

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.

To indent a paragraph from both the left and right margin of a page, do not set this property. Instead, add to the `Style` property of this paragraph a `mlreportgen.dom.OuterMargin` object specifying the left and right indentations.

Setting the `OuterLeftMargin` property adds a corresponding `mlreportGen.dom.OuterMargin` format object to the `Style` property for this document element. Removing the `OuterLeftMargin` property setting removes the object.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the indentation is expressed. Use one of these abbreviations for the units for indentation.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`OutlineLevel` — Outline level of this paragraph

[ ] (default) | numeric value

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 table of contents. For example, to make a paragraph appear as a `Heading 1` (Word) or `h1` (HTML), set `OutlineLevel` to `1`.

Data Types: `int32`

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Strike` — Text strikethrough

string

The default for this property is []. You can set it to one of these values:

- `none` — Do not use strikethrough.
- `single` — Use a single line for strikethrough.
- `double` — Use a double line for strikethrough for Word documents.

Setting the `Strike` property adds a corresponding `mlreportGen.dom.Strike` format object to the `Style` property for this document element. Removing the `Strike` property setting 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 stylesheet

string

Name of the style to apply from the stylesheet, specified as a string.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Underline` — Type of underline, if any, for text

[ ] (default) | string

You can specify one of the following types of underlines.

Border String	Description	Supported Output Types
`dash`	Dashed underline	Word
`dashedHeavy`	Line with heavy dashes	Word
`dashLong`	Line with long dashes	Word
`dashLongHeavy`	Line with heavy long dashes	Word
`dashDotDotHeavy`	Line with heavy dashes with two dots between the dashes	Word
`dashDotHeavy`	Heavy dash-dot line	Word
`dotted`	Dotted line	Word
`dottedHeavy`	Thick dotted line	Word
`dotDash`	Dot-dash line	Word
`dotDotDash`	Dot-dot-dash line	Word
`dashDotHeavy`	Heavy dot-dash line	Word
`double`	Double line	Word
`none`	Do not use underlining	All output types
`single`	Single line	All output types
`thick`	Thick line	Word
`wave`	Wavy line	Word
`waveyDouble`	Double wavy line	Word
`waveyHeavy`	Heavy wavy	Word
`words`	Underline non-space characters only	Word

If this property is empty and `StyleName` property of this document element specifies a style sheet style, the type of underline is determined by that style.

To specify the color as well as the type of the underline, do not set the `Underline` property. Instead, set the `Style` property of this document element to include an `mlreportgen.dom.Underline` format object that specifies the desired underline type and color.

Setting the `Underline` property adds a corresponding `mlreportgen.dom.Underline` format object to the `Style` property for this document element. Removing the `Underline` property setting removes the object.

`WhiteSpace` — White space and line breaks in text

[ ] (default) | string

To specify how to handle white space, use one of these values.

String	Description	Supported Output Types
`normal` (default)	For HTML and PDF, removes spaces at beginning and end of text string. Multiple spaces within string collapse to single space. For Word, removes spaces at beginning and end of text string.	All output types
`nowrap`	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
`preserve`	Preserves spaces and line feeds. Acts like the `<pre>` tag in HTML.	All output types
`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.	HTML and PDF

String	Description	Supported Output Types
`pre-wrap`	Preserves white space. Text wraps when necessary and on line breaks	HTML and PDF

Setting the `WhiteSpace` property adds a corresponding `WhiteSpace` format object to `Style` property. Removing the `WhiteSpace` property setting removes the `WhiteSpace` object.

Methods

Methods

Method	Purpose
append Use `Heading3.append` similar to how you use `Paragraph.append`.	Append content to heading.
clone Use `Heading3.clone` similar to how you use `Paragraph.clone`.	Copy heading.

Examples

Create Three Levels of Headings

This example shows how to add headings that are formatted according to their heading 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.Heading | mlreportgen.dom.Heading1 | mlreportgen.dom.Heading2 | mlreportgen.dom.Heading4 | mlreportgen.dom.Heading5 | mlreportgen.dom.Heading6 | mlreportgen.dom.Paragraph

More About

- “Report Formatting Approaches” on page 13-21

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

string

Heading text, specified as a string.

`styleName` — Style for the heading

string

The name of a style, specified as a string. 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

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`Bold` — Option to use bold for text

[ ] (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 text 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`

`Children` — Children of this object

cell array of objects

This read-only property lists child elements of this object.

`Color` — Text color

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`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

string

Amount by which to indent the first line of this paragraph relative to succeeding lines.

To create a hanging indent, in which all the lines are indented except for the first line, use a negative number.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the indentation is expressed. Use one of these abbreviations for the units for indentation.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`FontFamilyName` — Name of font family for text

string

The name of a font family.

To specify substitutions for this font, do not set this property. Instead create and add a `mlreportgen.dom.FontFamily` object to the `Style` property of this document element.

Setting the `FontFamilyName` property adds a corresponding `mlreportGen.dom.FontFamily` format object to the `Style` property for this document element. Removing the `FontFamilyName` property setting removes the object.

`FontSize` — Font size for text

string

If you need to specify substitutions for this font, do not set this property. Instead create and add a `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.

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the font size is expressed. Use one of these abbreviations for the units for the font size.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Italic` — Option to use italics for text

[ ] (default) | logical value

To use italics for text, 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 text 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`

`OuterLeftMargin` — Left indentation for paragraph

string

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.

To indent a paragraph from both the left and right margin of a page, do not set this property. Instead, add to the `Style` property of this paragraph a `mlreportgen.dom.OuterMargin` object specifying the left and right indentations.

Setting the `OuterLeftMargin` property adds a corresponding `mlreportGen.dom.OuterMargin` format object to the `Style` property for this document element. Removing the `OuterLeftMargin` property setting removes the object.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the indentation is expressed. Use one of these abbreviations for the units for indentation.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`OutlineLevel` — Outline level of this paragraph

[ ] (default) | numeric value

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 table of contents. For example, to make a paragraph appear as a `Heading 1` (Word) or `h1` (HTML), set `OutlineLevel` to `1`.

Data Types: `int32`

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Strike` — Text strikethrough

string

The default for this property is []. You can set it to one of these values:

- `none` — Do not use strikethrough.
- `single` — Use a single line for strikethrough.
- `double` — Use a double line for strikethrough for Word documents.

Setting the `Strike` property adds a corresponding `mlreportGen.dom.Strike` format object to the `Style` property for this document element. Removing the `Strike` property setting 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 stylesheet

string

Name of the style to apply from the stylesheet, specified as a string.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Underline` — Type of underline, if any, for text

[ ] (default) | string

You can specify one of the following types of underlines.

Border String	Description	Supported Output Types
`dash`	Dashed underline	Word
`dashedHeavy`	Line with heavy dashes	Word
`dashLong`	Line with long dashes	Word
`dashLongHeavy`	Line with heavy long dashes	Word
`dashDotDotHeavy`	Line with heavy dashes with two dots between the dashes	Word
`dashDotHeavy`	Heavy dash-dot line	Word
`dotted`	Dotted line	Word
`dottedHeavy`	Thick dotted line	Word
`dotDash`	Dot-dash line	Word
`dotDotDash`	Dot-dot-dash line	Word
`dashDotHeavy`	Heavy dot-dash line	Word
`double`	Double line	Word
`none`	Do not use underlining	All output types
`single`	Single line	All output types
`thick`	Thick line	Word
`wave`	Wavy line	Word
`waveyDouble`	Double wavy line	Word
`waveyHeavy`	Heavy wavy	Word
`words`	Underline non-space characters only	Word

If this property is empty and `StyleName` property of this document element specifies a style sheet style, the type of underline is determined by that style.

To specify the color as well as the type of the underline, do not set the `Underline` property. Instead, set the `Style` property of this document element to include an `mlreportgen.dom.Underline` format object that specifies the desired underline type and color.

Setting the `Underline` property adds a corresponding `mlreportgen.dom.Underline` format object to the `Style` property for this document element. Removing the `Underline` property setting removes the object.

`WhiteSpace` — White space and line breaks in text

[ ] (default) | string

To specify how to handle white space, use one of these values.

String	Description	Supported Output Types
`normal` (default)	For HTML and PDF, removes spaces at beginning and end of text string. Multiple spaces within string collapse to single space. For Word, removes spaces at beginning and end of text string.	All output types
`nowrap`	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
`preserve`	Preserves spaces and line feeds. Acts like the `<pre>` tag in HTML.	All output types
`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.	HTML and PDF

String	Description	Supported Output Types
`pre-wrap`	Preserves white space. Text wraps when necessary and on line breaks	HTML and PDF

Setting the `WhiteSpace` property adds a corresponding `WhiteSpace` format object to `Style` property. Removing the `WhiteSpace` property setting removes the `WhiteSpace` object.

Methods

Method	Purpose
append Use `Heading4.append` similar to how you use `Paragraph.append`.	Append content to heading.
clone Use `Heading4.clone` similar to how you use `Paragraph.clone`.	Copy heading.

Examples

Create Three Levels of Headings

This example shows how to add headings that are formatted according to their heading 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.Heading | mlreportgen.dom.Heading1 | mlreportgen.dom.Heading2 | mlreportgen.dom.Heading3 | mlreportgen.dom.Heading5 | mlreportgen.dom.Heading6 | mlreportgen.dom.Paragraph

More About

- “Report Formatting Approaches” on page 13-21

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

string

Heading text, specified as a string.

`styleName` — Style for the heading

string

The name of a style, specified as a string. 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

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`Bold` — Option to use bold for text

[ ] (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 text 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`

`Children` — Children of this object

cell array of objects

This read-only property lists child elements of this object.

`Color` — Text color

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`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

string

Amount by which to indent the first line of this paragraph relative to succeeding lines.

To create a hanging indent, in which all the lines are indented except for the first line, use a negative number.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the indentation is expressed. Use one of these abbreviations for the units for indentation.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`FontFamilyName` — Name of font family for text

string

The name of a font family.

To specify substitutions for this font, do not set this property. Instead create and add a `mlreportgen.dom.FontFamily` object to the `Style` property of this document element.

Setting the `FontFamilyName` property adds a corresponding `mlreportGen.dom.FontFamily` format object to the `Style` property for this document element. Removing the `FontFamilyName` property setting removes the object.

`FontSize` — Font size for text

string

If you need to specify substitutions for this font, do not set this property. Instead create and add a `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.

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the font size is expressed. Use one of these abbreviations for the units for the font size.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Italic` — Option to use italics for text

[ ] (default) | logical value

To use italics for text, 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 text 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`

`OuterLeftMargin` — Left indentation for paragraph

string

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.

To indent a paragraph from both the left and right margin of a page, do not set this property. Instead, add to the `Style` property of this paragraph a `mlreportgen.dom.OuterMargin` object specifying the left and right indentations.

Setting the `OuterLeftMargin` property adds a corresponding `mlreportGen.dom.OuterMargin` format object to the `Style` property for this document element. Removing the `OuterLeftMargin` property setting removes the object.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the indentation is expressed. Use one of these abbreviations for the units for indentation.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`OutlineLevel` — Outline level of this paragraph

[ ] (default) | numeric value

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 table of contents. For example, to make a paragraph appear as a `Heading 1` (Word) or `h1` (HTML), set `OutlineLevel` to `1`.

Data Types: `int32`

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Strike` — Text strikethrough

string

The default for this property is []. You can set it to one of these values:

- `none` — Do not use strikethrough.
- `single` — Use a single line for strikethrough.
- `double` — Use a double line for strikethrough for Word documents.

Setting the `Strike` property adds a corresponding `mlreportGen.dom.Strike` format object to the `Style` property for this document element. Removing the `Strike` property setting 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 stylesheet

string

Name of the style to apply from the stylesheet, specified as a string.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Underline` — Type of underline, if any, for text

[ ] (default) | string

You can specify one of the following types of underlines.

Border String	Description	Supported Output Types
`dash`	Dashed underline	Word
`dashedHeavy`	Line with heavy dashes	Word
`dashLong`	Line with long dashes	Word
`dashLongHeavy`	Line with heavy long dashes	Word
`dashDotDotHeavy`	Line with heavy dashes with two dots between the dashes	Word
`dashDotHeavy`	Heavy dash-dot line	Word
`dotted`	Dotted line	Word
`dottedHeavy`	Thick dotted line	Word
`dotDash`	Dot-dash line	Word
`dotDotDash`	Dot-dot-dash line	Word
`dashDotHeavy`	Heavy dot-dash line	Word
`double`	Double line	Word
`none`	Do not use underlining	All output types
`single`	Single line	All output types
`thick`	Thick line	Word
`wave`	Wavy line	Word
`waveyDouble`	Double wavy line	Word
`waveyHeavy`	Heavy wavy	Word
`words`	Underline non-space characters only	Word

If this property is empty and `StyleName` property of this document element specifies a style sheet style, the type of underline is determined by that style.

To specify the color as well as the type of the underline, do not set the `Underline` property. Instead, set the `Style` property of this document element to include an `mlreportgen.dom.Underline` format object that specifies the desired underline type and color.

Setting the `Underline` property adds a corresponding `mlreportgen.dom.Underline` format object to the `Style` property for this document element. Removing the `Underline` property setting removes the object.

`WhiteSpace` — White space and line breaks in text

[ ] (default) | string

To specify how to handle white space, use one of these values.

String	Description	Supported Output Types
`normal` (default)	For HTML and PDF, removes spaces at beginning and end of text string. Multiple spaces within string collapse to single space. For Word, removes spaces at beginning and end of text string.	All output types
`nowrap`	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
`preserve`	Preserves spaces and line feeds. Acts like the `<pre>` tag in HTML.	All output types
`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.	HTML and PDF

String	Description	Supported Output Types
`pre-wrap`	Preserves white space. Text wraps when necessary and on line breaks	HTML and PDF

Setting the `WhiteSpace` property adds a corresponding `WhiteSpace` format object to `Style` property. Removing the `WhiteSpace` property setting removes the `WhiteSpace` object.

Methods

Method	Purpose
append Use `Heading5.append` similar to how you use `Paragraph.append`.	Append content to heading.
clone Use `Heading5.clone` similar to how you use `Paragraph.clone`.	Copy heading.

Examples

Create Three Levels of Headings

This example shows how to add headings that are formatted according to their heading 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.Heading | mlreportgen.dom.Heading1 | mlreportgen.dom.Heading2 | mlreportgen.dom.Heading3 | mlreportgen.dom.Heading4 | mlreportgen.dom.Heading6 | mlreportgen.dom.Paragraph

More About

- “Report Formatting Approaches” on page 13-21

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

string

Heading text, specified as a string.

`styleName` — Style for the heading

string

The name of a style, specified as a string. 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

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`Bold` — Option to use bold for text

[ ] (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 text 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`

`Children` — Children of this object

cell array of objects

This read-only property lists child elements of this object.

`Color` — Text color

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`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

string

Amount by which to indent the first line of this paragraph relative to succeeding lines.

To create a hanging indent, in which all the lines are indented except for the first line, use a negative number.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the indentation is expressed. Use one of these abbreviations for the units for indentation.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`FontFamilyName` — Name of font family for text

string

The name of a font family.

To specify substitutions for this font, do not set this property. Instead create and add a `mlreportgen.dom.FontFamily` object to the `Style` property of this document element.

Setting the `FontFamilyName` property adds a corresponding `mlreportGen.dom.FontFamily` format object to the `Style` property for this document element. Removing the `FontFamilyName` property setting removes the object.

`FontSize` — Font size for text

string

If you need to specify substitutions for this font, do not set this property. Instead create and add a `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.

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the font size is expressed. Use one of these abbreviations for the units for the font size.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Italic` — Option to use italics for text

[ ] (default) | logical value

To use italics for text, 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 text 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`

`OuterLeftMargin` — Left indentation for paragraph

string

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.

To indent a paragraph from both the left and right margin of a page, do not set this property. Instead, add to the `Style` property of this paragraph a `mlreportgen.dom.OuterMargin` object specifying the left and right indentations.

Setting the `OuterLeftMargin` property adds a corresponding `mlreportGen.dom.OuterMargin` format object to the `Style` property for this document element. Removing the `OuterLeftMargin` property setting removes the object.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the indentation is expressed. Use one of these abbreviations for the units for indentation.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`OutlineLevel` — Outline level of this paragraph

[ ] (default) | numeric value

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 table of contents. For example, to make a paragraph appear as a `Heading 1` (Word) or `h1` (HTML), set `OutlineLevel` to `1`.

Data Types: `int32`

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Strike` — Text strikethrough

string

The default for this property is []. You can set it to one of these values:

- `none` — Do not use strikethrough.
- `single` — Use a single line for strikethrough.
- `double` — Use a double line for strikethrough for Word documents.

Setting the `Strike` property adds a corresponding `mlreportGen.dom.Strike` format object to the `Style` property for this document element. Removing the `Strike` property setting 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 stylesheet

string

Name of the style to apply from the stylesheet, specified as a string.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Underline` — Type of underline, if any, for text

[ ] (default) | string

You can specify one of the following types of underlines.

Border String	Description	Supported Output Types
`dash`	Dashed underline	Word
`dashedHeavy`	Line with heavy dashes	Word
`dashLong`	Line with long dashes	Word
`dashLongHeavy`	Line with heavy long dashes	Word
`dashDotDotHeavy`	Line with heavy dashes with two dots between the dashes	Word
`dashDotHeavy`	Heavy dash-dot line	Word
`dotted`	Dotted line	Word
`dottedHeavy`	Thick dotted line	Word
`dotDash`	Dot-dash line	Word
`dotDotDash`	Dot-dot-dash line	Word
`dashDotHeavy`	Heavy dot-dash line	Word
`double`	Double line	Word
`none`	Do not use underlining	All output types
`single`	Single line	All output types
`thick`	Thick line	Word
`wave`	Wavy line	Word
`waveyDouble`	Double wavy line	Word
`waveyHeavy`	Heavy wavy	Word
`words`	Underline non-space characters only	Word

If this property is empty and `StyleName` property of this document element specifies a style sheet style, the type of underline is determined by that style.

To specify the color as well as the type of the underline, do not set the `Underline` property. Instead, set the `Style` property of this document element to include an `mlreportgen.dom.Underline` format object that specifies the desired underline type and color.

Setting the `Underline` property adds a corresponding `mlreportgen.dom.Underline` format object to the `Style` property for this document element. Removing the `Underline` property setting removes the object.

`WhiteSpace` — White space and line breaks in text

[ ] (default) | string

To specify how to handle white space, use one of these values.

String	Description	Supported Output Types
`normal` (default)	For HTML and PDF, removes spaces at beginning and end of text string. Multiple spaces within string collapse to single space. For Word, removes spaces at beginning and end of text string.	All output types
`nowrap`	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
`preserve`	Preserves spaces and line feeds. Acts like the `<pre>` tag in HTML.	All output types
`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.	HTML and PDF

String	Description	Supported Output Types
`pre-wrap`	Preserves white space. Text wraps when necessary and on line breaks	HTML and PDF

Setting the `WhiteSpace` property adds a corresponding `WhiteSpace` format object to `Style` property. Removing the `WhiteSpace` property setting removes the `WhiteSpace` object.

Methods

Method	Purpose
append Use `Heading6.append` similar to how you use `Paragraph.append`.	Append content to heading.
clone Use `Heading6.clone` similar to how you use `Paragraph.clone`.	Copy heading.

Examples

Create Three Levels of Headings

This example shows how to add headings that are formatted according to their heading 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.Heading | mlreportgen.dom.Heading1 | mlreportgen.dom.Heading2 | mlreportgen.dom.Heading3 | mlreportgen.dom.Heading4 | mlreportgen.dom.Heading5 | mlreportgen.dom.Paragraph

More About

- “Report Formatting Approaches” on page 13-21

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
`'1in'` (default) | string

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the height is expressed. The following abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

Output Arguments

`heightObj` — Height of object
`mlreportgen.dom.Height` object

Height of object, represented by an `mlreportgen.dom.Height` object.

Properties

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Object height

`1in` (default) | string

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the height is expressed. The following abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

See Also

mlreportgen.dom.RowHeight | mlreportgen.dom.Width

More About

- "Report Formatting Approaches" on page 13-21

mlreportgen.dom.HTML class

Package: mlreportgen.dom

Convert HTML text to container of DOM objects

Description

Converts a string of HTML text to an `HTML` object containing DOM objects having the same content and format.

Construction

`htmlObj = HTML()` creates an empty `HTML` object.

`htmlObj = HTML(htmlText)` converts a string of 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 appear only in HTML output. Report generation ignores custom CSS properties for Microsoft Word and PDF output.

HTML Element	Attributes
`a`	`class`, `style`, `href`, `name`
`b`	`class`, `style`
`body`	`class`, `style`
`br`	n/a
`code`	`class`, `style`
`del`	`class`, `style`
`div`	`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, alt
li	class, style
ol	class, style
p	class, style, align
pre	class, style
s	class, style
span	class, style
strike	class, style
sub	class, style
sup	class, style
table	class, style, align, bgcolor, border, cellspacing, cellpadding, 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, valign, nowrap
tr	class, style, bgcolor, valign
tt	class, style
u	class, style
ul	class, style

For information about these elements, see the W3Schools tags documentation at www.w3schools.com/tags.

These CSS formats are supported:

- `background-color`

- border
- border-bottom
- border-bottom-color
- border-bottom-style
- boder-bottom-width
- border-color
- border-left
- border-left-color
- border-left-style
- boder-left-width
- border-right
- border-right-color
- border-rigtht-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, see the W3Schools CSS documentation at www.w3schools.com/cssref.

Input Arguments

`htmlText` — HTML text

string

HTML text, specified as a string

Example: `html = HTML('<p><b>Hello</b> <i style="color:green">World</i></p>');`

Properties

`Id` — ID for `HTML` object

string

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) | string

Tag name of HTML container element, specified as a string, 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

string

Style name of this HTML object, specified as a string. Use a name of a style specified in the stylesheet 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

string

Tag for HTML object, specified as a string.

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.

Methods

Examples

Convert HTML Text to a Word Report

Create an HTML object from HTML text to use for a Word report.

```
import mlreportgen.dom.*;
rpt = Document('MyRep1','docx');
html = HTML('<p><b>Hello</b> <i style="color:green">World</i></p>');
```

Append content to the HTML object and append the HTML object to the document.

```
append(html,'<p>This is <u>me</u> speaking</p>');
append(rpt,html);
```

Generate the Word report.

```
close(rpt);
rptview(rpt.OutputPath);
```

- “Append HTML Content to DOM Reports” on page 13-105

See Also

mlreportgen.dom.Document.addHTML | mlreportgen.dom.HTMLFile

More About

- “Appending HTML to DOM Reports” on page 13-103
- “HTML Code Requirements for DOM Reports” on page 13-113

External Websites

- www.w3schools.com/tags
- www.w3schools.com/cssref

Introduced in R2015a

mlreportgen.dom.HTMLFile class

Package: mlreportgen.dom

Convert contents of HTML file to container of DOM objects

Description

Converts the contents of an HTML file to an `HTMLFile` object containing DOM objects having the same content and format.

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 appear only in HTML output. Report generation ignores custom CSS properties for Microsoft Word and PDF output.

HTML Element	Attributes
`a`	`class`, `style`, `href`, `name`
`b`	`class`, `style`
`body`	`class`, `style`
`br`	n/a
`code`	`class`, `style`
`del`	`class`, `style`
`div`	`class`, `style`
`font`	`class`, `style`, `color`, `face`, `size`
`h1, h2, h3, h4, h5, h6`	`class`, `style`, `align`
`hr`	`class`, `style`, `align`

HTML Element	Attributes
i	class, style
ins	class, style
img	class, style, src, height, width, alt
li	class, style
ol	class, style
p	class, style, align
pre	class, style
s	class, style
span	class, style
strike	class, style
sub	class, style
sup	class, style
table	class, style, align, bgcolor, border, cellspacing, cellpadding, 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, valign, nowrap
tr	class, style, bgcolor, valign
tt	class, style
u	class, style
ul	class, style

For information about these elements, see the W3Schools tags documentation at www.w3schools.com/tags.

These CSS formats are supported:

- `background-color`
- `border`

- border-bottom
- border-bottom-color
- border-bottom-style
- boder-bottom-width
- border-color
- border-left
- border-left-color
- border-left-style
- boder-left-width
- border-right
- border-right-color
- border-rigtht-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, see the W3Schools CSS documentation at www.w3schools.com/cssref.

Input Arguments

`htmlFile` — HTML file path

string

HTML file path, specified as a string.

Properties

`Id` — ID for `HTMLFile` object

string

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) | string

Tag name of HTML container element, specified as a string, 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

string

Style name of this `HTMLFile` object, specified as a string. Use a name of a style specified in the stylesheet 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

string

Tag for `HTMLFile` object, specified as a string.

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.

Methods

Examples

Convert HTML File to Word Report

This example assumes that there are HTML files called `myHTMLfile1.html` and `myHTMLfile2.html` in the MATLAB current folder.

Create a Microsoft Word report.

```
import mlreportgen.dom.*;
rpt = Document('MyHTMLReport','docx');
```

Create the first `HTMLFile` object and append HTML markup and an `HTML` object to the `HTMLFile` object.

```
path = 'myHTMLfile1.html';
htmlFile1 = HTMLFile(path);
append(htmlFile1,'<p>This is <u>HTML markup text</u></p>');
htmlObj = HTML('<p>This is an <b>HTML object</b></p>');
append(htmlFile1,htmlObj);
```

Create a second `HTMLFile` object.

```
htmlFile2 = HTMLFile('myHTMLFile2.html');
```

Append the second `HTMLFile` object to the first `HTMLFile` object, and append the first `HTMLFile` object to the document.

```
append(htmlFile1,htmlFile2)
append(rpt,htmlFile1);
```

Generate the report.

```
close(rpt);
rptview(rpt.OutputPath);
```

- “Append HTML File Contents to DOM Reports” on page 13-107

See Also

mlreportgen.dom.Document.addHTMLFile | mlreportgen.dom.HTML

More About

- “Appending HTML to DOM Reports” on page 13-103
- “HTML Code Requirements for DOM Reports” on page 13-113

External Websites

- www.w3schools.com/tags
- www.w3schools.com/cssref

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
string

Line style for horizontal rule, specified as a string. Use one of these values.

String	Applies To	
	DOCX	HTML and PDF
'dashed'	✓	✓
'dashdotstroked'	✓	
'dashsmallgap'	✓	
'dotted'	✓	✓
'dotdash'	✓	
'dotdotdash'	✓	
'double'	✓	✓
'doublewave'	✓	
'inset'	✓	✓
'none'	✓	✓
'outset'	✓	✓
'single'	✓	
'solid'		✓
'thick'	✓	
'thickthinlargegap'	✓	
'thickthinmediumgap'	✓	
'thickthinsmallgap'	✓	
'thinthicklargegap'	✓	

String	Applies To	
	DOCX	HTML and PDF
'thinthickmediumgap'	✓	
'thinthicksmallgap'	✓	
'thinthickthinlargegap'	✓	
'thinthickthinmediumgap'	✓	
'thinthickthinsmallgap'	✓	
'threedemboss'	✓	
'threedengrave'	✓	
'triple'	✓	
'wave'	✓	

`BorderColor` — Color of line

string

Color of the line, specified as a string. You can specify:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`BorderWidth` — Width of line (in HTML report)

string

Width of line (in an HTML report), specified as a string. The string must have the format `valueUnits` where `Units` is an abbreviation for the width. Valid abbreviations are:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches

- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixel

`BackgroundColor` — Background color of line

string

Background color of the line, specified as a string. You can specify:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`HAlign` — Horizontal alignment of line

string

Horizontal alignment of the line, specified as a string.

String	Description	Applies To	
		DOCX	HTML and PDF
`'center'`	Align center	✓	✓
`'distribute'`	Distribute all characters equally	✓	
`'justify'`	Justified		✓
`'KashidaHigh'`	Widest Kashida length	✓	
`'Kashida Low'`	Low Kashida length	✓	
`'KashidaMedium'`	Medium Kashida length	✓	
`'left'`	Align left	✓	✓
`'right'`	Align right	✓	✓

String	Description	Applies To	
		DOCX	HTML and PDF
`'ThaiDistribute'`	Thai language distribution	✓	

Note: Kashida is a type of justification used for some cursive scripts, primarily Arabic, and Persian.

`OuterLeftMargin` — Outer left margin (indentation) of line

string

Outer left margin (indentation) of line, specified as a string. The string must have the format `valueUnits` where `Units` is an abbreviation for the margin. Valid abbreviations are:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`OuterRightMargin` — Outer right margin (indentation) of line

string

Outer right margin (indentation) of line, specified as a string. The string must have the format `valueUnits` where `Units` is an abbreviation for the margin. Valid abbreviations are:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas

- `pt` — points
- `px` — pixels

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Style` — Format specification for line

array of format objects

Format specification for the line, specified as an array of format objects.

`StyleName` — Stylesheet style for line

string

Stylesheet style for line, specified as a string. The name of a style must be specified in the stylesheet 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

string

Tag for line, specified as a string.

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 Use `HorizontalRule.clone` in a similar way to how you use `Paragraph.clone`.	Copy horizontal line.

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);
```

Introduced in R2015b

mlreportgen.dom.Image class

Package: mlreportgen.dom

Create image to include in report

Description

Create an image to include in a report.

Construction

`imageObj = Image(imagePath)` creates an image object containing the image file specified by `imagePath`.

The contents of the specified image file are copied into the output document either when the image object is appended to the document (in document streaming mode) or when the document is closed. Do not delete the original file before it has been copied into the document.

Input Arguments

`imagePath` — Path of image file

string

Path of an image file, specified as a string. You can use these image formats.

Image Format	Supported Output Formats		
	HTML	Word	PDF
`.bmp`	✓	✓	✓
`.emf`		✓	
`.gif`	✓	✓	✓
`.jpg`	✓	✓	✓

Image Format	Supported Output Formats		
	HTML	Word	PDF
`.png`	✓	✓	✓
`.svg`	✓		✓
`.tif`		✓	✓

Output Arguments

`imageObj` — Image
`mlreportgen.dom.Image` object

Image, represented by an `mlreportgen.dom.Image` object.

Properties

`CustomAttributes` — Custom attributes of document element
array of `mlreportgen.dom.CustomAttribute` objects

The output format must support the custom attributes of this document element.

`Height` — Height of image
string

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the height is expressed. The following abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

Alternatively, you can specify the image height using the `Image.Style` property. For example:

```
Image.Style = {Height('4in')};
```

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Map` — Map of hyperlink areas in image (HTML only)

`mlreportgen.dom.ImageMap` object

Map of hyperlink areas in image, specified as an `mlreportgen.dom.ImageMap` object

`Path` — Path of image file

string

Path of image file, specified as a string.

`Style` — Format specification

array of format objects

Format objects that specify the format of a document element.

`StyleName` — Name of image style

string

Name of image style, specified as a string. The style name is the name of a style specified in the stylesheet of the document or document part to which this image is appended. The specified style defines the appearance of this image 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 document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`VAlign` — Vertical alignment of image

string

Possible values are:

- `top`
- `bottom`
- `middle`

`Width` — Image width

string

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the height is expressed. The following abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

Methods

Method	Purpose
`append` Use `Image.append` in a similar way to how you use `ExternalLink.append`.	Append a custom element to this image.

Method	Purpose
`clone` Use `Image.clone` in a similar way to how you use `Paragraph.clone`.	Clone this image

See Also

mlreportgen.dom.ImageArea | mlreportgen.dom.ImageMap | mlreportgen.dom.ScaleToFit

Related Examples

- “Create and Format Images” on page 13-86

mlreportgen.dom.ImageArea class

Package: mlreportgen.dom

Define image area as hyperlink

Description

Define an area in an image to be an HTML hyperlink. When a user clicks an image area, the HTML browser displays the target page, based on the URL or link target you specify. You can provide alternative text for screen reader users.

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

string

String that specifies either of these values:

- 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

string

Text to display if the image is not visible, specified as a string.

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

string

String that specifies either of these values:

- 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

string

Text to display if the image is not visible, specified as a string.

`Shape` — Shape of image area

string

(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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Examples

Add Image Area to Image of a MATLAB Plot

```
import mlreportgen.dom.*
d = Document('imageArea','html');
```

```
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');

plot1 = Image('plot_img.png');
append(d,plot1);

area1 = ImageArea(...
    ['http://www.mathworks.com/help/matlab/ref/' ...
    'plot.html?searchHighlight=plot'], ...
    'plot function help', 240,450,463,492);

map = ImageMap();
plot1.Map = map;
append(map,area1);

close(d);
rptview(d.OutputPath);
```

- “Create and Format Images” on page 13-86

See Also

mlreportgen.dom.Image | mlreportgen.dom.ImageMap | mlreportgen.dom.LinkTarget

mlreportgen.dom.ImageMap class

Package: mlreportgen.dom

Map of hyperlink areas in image

Description

Map of image areas, which are areas within an image where a user can click to open content in an HTML browser.

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, specified as an `mlreportgen.dom.ImageMap` object.

Properties

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Methods

Method	Purpose
`append`	Append an image area to this image map.
`clone` Use `ImageMap.clone` in a similar way you how you use `Paragraph.clone`.	Clone this image map.

Examples

Append an Image Area to an Image Map

```
import mlreportgen.dom.*
d = Document('imageArea','html');

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');

plot1 = Image('plot_img.png');
append(d,plot1);

area1 = ImageArea(...
    ['http://www.mathworks.com/help/matlab/ref/' ...
    'plot.html?searchHighlight=plot'], ...
    'plot function help', 240,450,463,492);

map = ImageMap();
```

```
plot1.Map = map;
append(map,area1);

close(d);
rptview(d.OutputPath);
```

- “Create and Format Images” on page 13-86

See Also

mlreportgen.dom.Image | mlreportgen.dom.ImageArea

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

string

String specifying the margin on all sides between the content of an object and its bounding box. String 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
- `pi` — picas
- `pt` — points
- `px` — pixel

`left` — Left margin size
string

String specifying the left margin between the content of an object and its bounding box. See the `all` input argument description for valid strings.

`right` — Right margin size
string

String specifying the right margin between the content of an object and its bounding box. See the `all` input argument description for valid strings.

`top` — Top margin size
string

String specifying the top margin between the content of an object and its bounding box. See the `all` input argument description for valid strings.

`bottom` — Bottom margin size
string

String specifying the bottom margin between the content of an object and its bounding box. See the `all` input argument description for valid strings.

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

string

String specifying the bottom margin. String 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
- `pi` — picas
- `pt` — points
- `px` — pixel

`Left` — Size of left margin

string

String specifying the left margin between the content of an object and its bounding box. See the `Bottom` property description for valid strings.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Right` — Size of right margin

string

String specifying the right margin between the content of an object and its bounding box. See the `Bottom` property description for valid strings.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Top` — Size of top margin

string

String specifying the top margin between the content of an object and its bounding box. See the `Bottom` property description for valid strings.

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

More About

- “Report Formatting Approaches” on page 13-21

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
string

Link target name, specified as string. The string is the `value in the Name property of an mlreportgen.dom.LinkTarget object` or a URL.

`linkText` — Link text
string

The text to use for the link text.

`linkTextStyleName` — Name of style for link text
string

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 this object
cell array of objects

This read-only property lists child elements of this object.

`CustomAttributes` — Custom attributes of document element
array of `mlreportgen.dom.CustomAttribute` objects

The output format must support the custom attributes of this document element.

`Id` — ID for document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Parent` — Parent of document element
DOM object

This read-only property lists the parent of this document element.

`StyleName` — Name of link style defined in the template
string

Name of link style defined in the template, specified as a string. 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 document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Target` — Internal target link

string

This read-only property displays the link target of this hyperlink.

Methods

Method	Purpose
append Use `InternalLink.append` in a similar way to how you use `ExternalLink.append`.	Append text or a `Text`, `Image`, or `CustomElement` object.
clone Use `InternalLink.clone` in a similar way to how you use `Paragraph.clone`.	Copy the internal link.

Examples

Add Internal Link

```
import mlreportgen.dom.*
d = Document('mydoc');

append(d, InternalLink('bio','About the Author'));
h = Heading(1,LinkTarget('bio'));
append(h,'Author''s Biography');
append(d,h);

close(d);
rptview('mydoc','html');
```

- “Create Links” on page 13-83

See Also

mlreportgen.dom.ExternalLink | mlreportgen.dom.LinkTarget

mlreportgen.dom.Italic class

Package: mlreportgen.dom

Italic for text object

Description

Specifies whether text should be rendered italic.

Construction

`italicObj = Italic()` creates a format object that specifies that text should be rendered italic.

`italicObj = Italic(value)` creates a format object that specifies that text should be rendered italic if `value` is `true`; otherwise, upright.

Input Arguments

`value` — Option to use italic or not for a text object

logical value

A setting of `false` (or `0`) uses upright text. A setting of `true` (or `1`) renders text in italic

Data Types: `logical`

Output Arguments

`italicObj` — Italic format object

`mlreportgen.dom.Italic` object

Italic format, represented by an `mlreportgen.dom.Italic` object.

Properties

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`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 that whose text is italic by default

```
import mlreportgen.dom.*
d = Document('mydoc');
p = Paragraph('italic text');
p.Style = {Italic};
append(d,p);
close(d);
rptview('mydoc', 'html');
```

Add Upright Text

```
import mlreportgen.dom.*
d = Document('mydoc');
p = Paragraph('italic text  ');
```

```
p.Style = {Italic};
append(d,p);
t = Text('upright text');
t.Style = {Italic(false)};
append(p,t);
close(d);
rptview('mydoc', 'html');
```

More About

- “Report Formatting Approaches” on page 13-21

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`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.KeepWithNext | mlreportgen.dom.PageBreakBefore

More About

- “Report Formatting Approaches” on page 13-21

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 document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element
string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`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.KeepLinesTogether | mlreportgen.dom.PageBreakBefore

More About

- "Report Formatting Approaches" on page 13-21

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 this object
cell array of objects

This read-only property lists child elements of this object.

`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 document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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

This read-only property lists the parent of this document element.

`Style` — Format specification
array of format objects

Format objects that specify the format of a document element.

`StyleName` — Style to apply from stylesheet
string

Name of the style to apply from the stylesheet, specified as a string.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, 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 this object.

Examples

Insert Leaders in a PDF Report

This example uses a dot leader in a PDF report. The example uses the dot leader and the space leader.

```
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
p2 = Paragraph();
append(p2,role);
append(p2,dotl);
append(p2,actor);
append(d,p2);

% Repeat, updating variables for each new paragraph
% Insert a clone of the dotl object
role = 'Juliet';
actor = 'Claire Danes';
p3 = Paragraph();
append(p3,role);
append(p3,clone(dotl));
append(p3,actor);
append(d,p3);

role = 'Tybalt';
actor = 'John Leguizamo';
p4 = Paragraph();
append(p4,role);
append(p4,clone(dotl));
append(p4,actor);
append(d,p4);

close(d);
rptview(d.OutputPath);
```

See Also

mlreportgen.dom.Heading1 | mlreportgen.dom.Paragraph | mlreportgen.dom.TOC

More About

- “Report Formatting Approaches” on page 13-21

Introduced in R2016a

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

string

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the height is expressed. The following abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters

- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points

`spacingType` — Line spacing type
string

Type of line spacing, specified as one of these strings:

- `'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.com.LineSpacing` object

Spacing between lines of paragraph, represented by an `mlreportgen.com.LineSpacing` object.

Properties

`Id` — ID for document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element
string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Type` — Option to specify type of line spacing height

string | `'exact'` |

Type of line spacing, specified as one of these strings:

- `'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

`'1in'` (default) | string

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the height is expressed. The following abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points

Examples

Create Line Spacing 1.5 Times the Height of the Paragraph Text Lines

```
p = Paragraph();
p.Style = {LineSpacing(1.5)};
```

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

string

Name of a link target, specified as a string.

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'/>`).

Microsoft Word replaces spaces in a link target names with underscore characters. Avoid spaces in link target names in Word reports.

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

The output format must support the custom attributes of this document element.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Name` — Name of link target

string

Microsoft Word replaces spaces in a link target names with underscore characters. Avoid spaces in link target names in Word reports.

`Style` — Format specification

array of format objects

Format objects that specify the format of a document element.

`Stylename` — Link target style name

string

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 document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Methods

Method	Purpose
append	Append content to link target.
clone Use `LinkTarget.clone` in a similar way to how you use `Paragraph.clone`.	Copy link target.

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. In an actual report, links to this target would appear further down in the report.

```
import mlreportgen.dom.*
d = Document('mydoc');

append(d,LinkTarget('home'));
append(d,InternalLink('home','Go to Top'));

close(d);
rptview('mydoc', 'html');
```

- “Create Links” on page 13-83

See Also

mlreportgen.dom.ExternalLink | mlreportgen.dom.ImageArea | mlreportgen.dom.InternalLink

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.

`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

string

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
string

Name of style to use for the list item, specified as a string.

Output Arguments

`listItemObj` — List item
`mlreportgen.dom.ListItem` object

List item, represented by an `mlreportgen.dom.ListItem` object.

Properties

`Children` — Children of this object
cell array of objects

This read-only property lists child elements of this object.

`CustomAttributes` — Custom attributes of document element
array of `mlreportgen.dom.CustomAttribute` objects

The output format must support the custom attributes of this document element.

`Id` — ID for document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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

This read-only property lists the parent of this document element.

`StyleName` — This property is ignored
string

This property is ignored.

`Tag` — Tag for document element
string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Methods

Method	Purpose
append Use `ListItem.append` in a similar way as you use `Paragraph.append`, except you can append different content to a list item than to a paragraph.	Append a string or any of these kinds of DOM objects to a list item: • `Text` • `Paragraph` • `Table` • `Image` • `ExternalLink` • `InternalLink` • `CustomElement`

Method	Purpose
clone Use `ListItem.clone` the same way you use `Paragraph.clone`.	Clone a list item.

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);
```

- “Create and Format Lists” on page 13-63

See Also

mlreportgen.dom.ListStyleType | mlreportgen.dom.OrderedList | mlreportgen.dom.UnorderedList

mlreportgen.dom.ListStyleType class

Package: mlreportgen.dom

Bullet or number style for HTML and PDF output

Description

Specifies the bullet or number style for an mlreportgen.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 8-40.

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) | string

Bullet or number style, specified as one of these strings.

Style value	Bullet or number type
`'disc'`	Filled circle (default)
`'armenian'`	Armenian numbering
`'circle'`	Open circle

Style value	Bullet or number type
`'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'`	Ktakana nnumbering
`'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 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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Bullet or number style

`'disc'` (default) | string

Bullet or number style, specified as one of these strings.

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
`'hiragana-iroha'`	Hiragana-iroha numbering
`'katakana'`	Ktakana nnumbering
`'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

Style value	Bullet or number type
`'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');
```

- “Create and Format Lists” on page 13-63

See Also

mlreportgen.dom.ListItem | mlreportgen.dom.OrderedList | mlreportgen.dom.UnorderedList

More About

- “Report Formatting Approaches” on page 13-21

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

string

(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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Methods

Method	Purpose
`dispatch`	Dispatch a document generation status message.
`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.

- "Display Report Generation Messages" on page 13-119

See Also

mlreportgen.dom.MessageDispatcher.dispatch | mlreportgen.dom.MessageDispatcher.getTheDispatcher | mlreportgen.dom.MessageEventData | mlreportgen.dom.MessageFilter

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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 document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

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);
```

- “Display Report Generation Messages” on page 13-119

See Also

`mlreportgen.dom.MessageDispatcher`

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

mlreportgen.dom.MessageDispatcher.dispatch

Related Examples

- “Display Report Generation Messages” on page 13-119

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 this object

cell array of objects

This read-only property lists child elements of this object.

`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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Style` — Format specification

array of format objects

Format objects that specify the format of a document element.

`StyleName` — Style to apply from stylesheet

string

Name of the style to apply from the stylesheet, specified as a string.

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 string 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.Paragraph | mlreportgen.dom.Text

More About

- "Report Formatting Approaches" on page 13-21

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

string

Name of part, specified as a string.

`sourcePath` — Path of source file for part

string

Path of source file for part, specified as a string.

Output Arguments

`opcPartObj` — OPC part

`mlreportgen.dom.OPCPart` object

OPC part, represented by an `mlreportgen.dom.OPCPart` object.

Properties

`ContentType` — Content type of part

string

Specifies the content type, using a file extension. For a list of file content types, see https://en.wikipedia.org/wiki/Open_Packaging_Conventions.

If you do not set this property is not set 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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Name` — Path of part

string

Path of this part relative to the root of the package, specified as a string. 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.

`RelatedPart` — Path name of part to which specified part is related

string

For information about OPC part names, see https://en.wikipedia.org/wiki/Open_Packaging_Conventions.

`RelationshipID` — Relationship ID

string

For information about OPC relationship IDs, see https://en.wikipedia.org/wiki/Open_Packaging_Conventions.

`RelationshipType` — Relationship type

string

Specifies a relationship type, using a file extension. For a list of file content types, see 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`

File Type	File Extension
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

string

Source file path, specified as a string.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

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 reportgen.dom.*;
d = Document('mydoc','html');
package(d,OPCPart('/data/data.json','data.json'));
close(d);
```

See Also

mlreportgen.dom.Document.package

More About

- “Report Packages” on page 13-17

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(listItems)` creates an ordered list from a cell array of strings specifying the list items.

Input Arguments

`listItems` — Content to include in the ordered list

cell array of strings

Cell array containing a combination of the following:

- A string
- 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 `listItems` argument.

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

The output format must support the custom attributes of this document element.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Style` — Format specification

array of format objects

Format objects that specify the format of a document element.

`Stylename` — List style name

string

The style specified by `styleName` must be defined in the template used to create the document element to which this list is appended.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Methods

Method	Purpose
append	Append items to this list.
clone Use the `OrderedList.clone` method similar to how you use `Paragraph.clone`.	Copy the list.

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');
```

- “Create and Format Lists” on page 13-63

See Also

mlreportgen.dom.ListItem | mlreportgen.dom.UnorderedList

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

string

String specifying the margin on all sides between the bounding box of an object and its surroundings. String 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`left` — Outer left margin size

string

String specifying the left margin between the bounding box of an object and its surroundings. See the `all` argument for description of valid strings.

`right` — Outer right margin size

string

String specifying the right margin between the bounding box of an object and its surroundings. See the `all` argument for description of valid strings.

`top` — Outer top margin size

string

String specifying the top margin between the bounding box of an object and its surroundings. See the `all` argument for description of valid strings.

`bottom` — Outer bottom margin size

string

String specifying the bottom margin between the bounding box of an object and its surroundings. See the `all` argument for description of valid strings.

Output Arguments

`marginObj` — Margin between bounding box and surroundings

`mlreportgen.dom.OuterMargin` object

A `mlreportgen.dom.OuterMargin` object specifying the margin between bounding box and surroundings.

Properties

`Bottom` — Size of bottom margin

string

String specifying the bottom margin. String 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`Left` — Size of left margin

string

String specifying the left margin. See the `Bottom` property for description of valid strings.

`Right` — Size of right margin

string

String specifying the right margin. See the `Bottom` property for description of valid strings.

`Top` — Size of top margin

string

String specifying the top margin. See the `Bottom` property for description of valid strings.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

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('Opt','Opt','Opt','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('Opt','Opt','Opt','12pt')}
append(d,p);

close(d);
rptview('test',doctype);
```

See Also

mlreportgen.dom.InnerMargin

More About

- "Report Formatting Approaches" on page 13-21

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`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.Heading | mlreportgen.dom.Paragraph

More About

- “Automatically Number Document Content” on page 13-98

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 this object

cell array of objects

This read-only property lists child elements of this object.

`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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Style` — Format specification

array of format objects

Format objects that specify the format of a document element.

`StyleName` — Style to apply from style sheet

string

Name of the style to apply from the style sheet, specified as a string.

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 ');
t.WhiteSpace = 'preserve';
pageinfo = Paragraph();
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);
```

- “Add Complex Page Numbers in Microsoft Word” on page 13-162

See Also

mlreportgen.dom.NumPages | mlreportgen.dom.PageNumber | mlreportgen.dom.Paragraph | mlreportgen.dom.Text

More About

- “Report Formatting Approaches” on page 13-21

Introduced in R2016a

mlreportgen.dom.PageBreak class

Package: mlreportgen.dom

Create page break object

Description

Creates a page break object that you can insert in a Microsoft Word or PDF report. Use `PageBreak` to insert a page break anywhere in a report.

Tip Use the `PageBreakBefore` format to force a page break before a specific paragraph. For example, use `PageBreakBefore` to force chapters to start on a new page.

Construction

`break = PageBreak()` creates a page break object.

Output Arguments

`break` — Page break

`mlreportgen.dom.PageBreak` object

Page break, returned as an `mlreportgen.dom.PageBreak` object.

Properties

`Children` — Children of this object

cell array of objects

This read-only property lists child elements of this object.

`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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Style` — Format specification

array of format objects

Format objects that specify the format of a document element.

`StyleName` — Style to apply from stylesheet

string

Name of the style to apply from the stylesheet, specified as a string.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, 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 this object.

Examples

Insert a Page Break

This example shows how to force a page break by inserting a `PageBreak` object into a PDF report.

```
import mlreportgen.dom.*;
d = Document('mydoc','pdf');
open(d);

% Create first page
h = Heading1('My First Head');
p = Paragraph('Here are some paragraphs.');
append(d,h);
append(d,p);
append(d,clone(p));
append(d,clone(p));
append(d,clone(p));
append(d,clone(p));
append(d,clone(p));

% Create and append the page break object
br = PageBreak();
append(d,br);

% Create paragraphs that appear on the page after the break
p2 = Paragraph('Here are some paragraphs after the forced page break.');
append(d,p2);
append(d,clone(p2));
append(d,clone(p2));
append(d,clone(p2));
append(d,clone(p2));

close(d);
rptview(d.OutputPath);
```

See Also

mlreportgen.dom.KeepLinesTogether | mlreportgen.dom.KeepWithNext | mlreportgen.dom.PageBreakBefore | mlreportgen.dom.Paragraph | mlreportgen.dom.WidowOrphanControl

More About

- “Report Formatting Approaches” on page 13-21

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.

Construction

`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`

Output Arguments

`pageBreakBefore` — Page break before format

`mlreportgen.dom.PageBreakBefore` object

Page break before format, returned as an `mlreportgen.dom.PageBreakBefore` object.

Properties

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`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

mlreportgen.dom.PageBreak | mlreportgen.dom.Paragraph

More About

- “Report Formatting Approaches” on page 13-21

mlreportgen.dom.PageMargins class

Package: mlreportgen.dom

Page margins for Microsoft Word and PDF page layout

Description

Specifies the size of the page margins of a section of a Microsoft Word or PDF document.

Word and PDF formats create page headers and footers differently. 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 format.

In addition, each format handles top and bottom margins differently. 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 header and the body text can overlap.
- 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 can overlap the body text.

Construction

`PageMarginsObj = PageMargins()` 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

`PageMarginsObj` — Page margins
`PageMargins` object

Page margins, returned as an `mlreportgen.dom.PageMargins` object.

Properties

`Bottom` — Bottom margin size

string

Width of the bottom margin, specified as a string. The string must use the format `valueUnits`, where `Units` is an abbreviation for the units to use. Valid abbreviations are:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Footer` — Footer size

string

Footer size, specified as a string. The string must use the format `valueUnits`, where `Units` is an abbreviation for the units to use. Valid abbreviations are:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Gutter` — Gutter size

string

Gutter size, specified as a string. The string must use the format `valueUnits`, where `Units` is an abbreviation for the units to use. Valid abbreviations are:

- no abbreviation — pixels

- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Header` — Header size

string

Header size, specified as a string. The string must use the format `valueUnits`, where `Units` is an abbreviation for the units to use. Valid abbreviations are:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Left` — Left margin size

string

Left margin size, specified as a string. The string must use the format `valueUnits`, where `Units` is an abbreviation for the units to use. Valid abbreviations are:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters

- `pi` — picas
- `pt` — points
- `px` — pixels

`Right` — Right margin size

string

Right margin size, specified as a string. The string must use the format `valueUnits`, where `Units` is an abbreviation for the units to use. Valid abbreviations are:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Top` — Top margin size

string

Top margin size, specified as a string. The string must use the format `valueUnits`, where `Units` is an abbreviation for the units to use. Valid abbreviations are:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches

- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

Examples

Reset Default Margins

Set the left and right margins for the current page layout to .5 inches.

```
import mlreportgen.dom.*;
d = Document('myreport','docx');
open(d);

s = d.CurrentPageLayout;
s.PageMargins.Left  = '.5in';
s.PageMargins.Right = '.5in';
append(d,'Left and right margins are .5 inch');

close(d);
rptview(d.OutputPath);
```

See Also

mlreportgen.dom.DOCXPageFooter | mlreportgen.dom.DOCXPageHeader | mlreportgen.dom.DOCXPageLayout | mlreportgen.dom.PDFPageFooter | mlreportgen.dom.PDFPageHeader | mlreportgen.dom.PDFPageLayout

More About

- “Report Formatting Approaches” on page 13-21

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 13-162.

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

`'a' | 'A' | 'i' | 'I' | 'n' | 'N'`

Type of page numbering to use, specified as one of these values:

- `'a'`— Lowercase alphabetic
- `'A'`— Uppercase alphabetic.
- `'i'`— Lowercase roman numerals.
- `'I'`— Uppercase roman numerals.
- `'n'` or `'N'`— Numeric page numbers.

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 string

Level of Heading style that chapters use, specified as a positive integer string.

`Format` — Type of page numbering to use

`'a' | 'A' | 'i' | 'I' | 'n' | 'N'`

Type of page numbering to use, specified as one of these values:

- `'a'`— Lowercase alphabetic

- `'A'` — Uppercase alphabetic.
- `'i'` — Lowercase roman numerals.
- `'I'` — Uppercase roman numerals.
- `'n'` or `'N'` — Numeric page numbers.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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 document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Examples

Style Numbers Using PageNumber

This example shows the syntax for using `PageNumber`. You set `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 13-162

```
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 = {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.OutputPath);
```

- “Add Complex Page Numbers in Microsoft Word” on page 13-162

See Also

mlreportgen.dom.DOCXPageFooter | mlreportgen.dom.DOCXPageLayout | mlreportgen.dom.Page | mlreportgen.dom.PDFPageFooter | mlreportgen.dom.PDFPageLayout

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Markup` — Word XML markup strings

cell array of strings

Specify a cell array of strings. Each string contains Word XML markup for a Word format.

For information about XML markup for Word formats, see http://officeopenxml.com/anatomyofOOXML.php.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

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

More About

- “Report Formatting Approaches” on page 13-21

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

string

Name of the link target to reference, specified as a string. 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 this object

cell array of objects

This read-only property lists child elements of this object.

`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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Style` — Format specification

array of format objects

Format objects that specify the format of a document element.

`StyleName` — Style to apply from stylesheet

string

Name of the style to apply from the stylesheet, specified as a string.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Target` — Name of target of this page reference

string

Name of the target of this page reference, specified as a string.

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);
```

- “Create Links” on page 13-83

See Also

mlreportgen.dom.LinkTarget | mlreportgen.dom.Page | mlreportgen.dom.Paragraph | mlreportgen.dom.StyleRef | mlreportgen.dom.Text

More About

- “Report Formatting Approaches” on page 13-21

Introduced in R2016a

mlreportgen.dom.PageSize class

Package: mlreportgen.dom

Size and orientation of pages in Microsoft Word and PDF documents

Description

Specifies the height, width, and orientation of pages in a section of a Microsoft Word or PDF document.

Construction

`PageSizeObj = PageSize()` creates a page size object having default values of 8.5-by-11 inches and portrait orientation.

`PageSizeObj = PageSize(height,width)` creates a portrait page having a specified height and width.

`PageSizeObj = PageSize(height,width,orientation)` creates a page having the specified height, width, and orientation.

Input Arguments

`height` — Height of page

`'11in'` (default) | string

Height of the page, specified as a string. The string must use the format `valueUnits`, where `Units` is one of these abbreviations:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`width` — Width of page

`'8.5in'` (default) | string

Width of page, specified as a string. The string must use the format `valueUnits`, where `Units` is one of these abbreviations:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`orientation` — Orientation of page

`'portrait'` (default) | `'landscape`

Orientation of the page, specified as:

- `'portrait'` for vertical orientation
- `'landscape'` for horizontal orientation

Specify height and width values that match the orientation setting. For example, if the orientation is landscape, for 8.5x11-inch paper, set `height` to `'8.5in'` and `width` to `'11in'`.

Output Arguments

`PageSizeObj` — Page size and orientation of document

`mlreportgen.dom.PageSize` object

Page size and orientation of te document, returned as an `mlreportgen.dom.PageSize` object.

Properties

`Height` — Height of pages in this section

`'11in'` (default) | string

Height of pages in this section, specified as a string. The string must use the format `valueUnits` where `Units` is one of these abbreviations:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Orientation` — Orientation of page

`'portrait'` (default) | `'landscape`

Orientation of the page, specified as:

- `'portrait'` for vertical orientation
- `'landscape'` for horizontal orientation

Specify height and width values that match the orientation setting. For example, if the orientation is landscape, for 8.5x11-inch paper, set `height` to `'8.5in'` and `width` to `'11in'`.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Width` — Width of page

`'8.5in'` (default) | string

Width of page, specified as a string. The string must use the format `valueUnits`, where `Units` is one of these abbreviations:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

Examples

Set Page Orientation and Size

Change the page orientation and size to landscape mode.

```
import mlreportgen.dom.*;
d = Document('myreport','docx');
open(d);

s = d.CurrentPageLayout;
s.PageSize.Orientation  ='landscape';
s.PageSize.Height = '8.5in';
s.PageSize.Width = '11in';
append(d,'This document has landscape pages');

close(d);
rptview(d.OutputPath);
```

- “Report Formatting Approaches” on page 13-21

See Also

mlreportgen.dom.DOCXPageLayout | mlreportgen.dom.PageMargins | mlreportgen.dom.PDFPageLayout

Introduced in R2016a

mlreportgen.dom.Paragraph class

Package: mlreportgen.dom

Formatted block of text (paragraph)

Description

Use a `mlreportgen.dom.Paragraph` object to define a paragraph. You can append document elements, such as an image, to a paragraph.

Construction

`paragraphObj = Paragraph(text)` creates a paragraph containing a `mlreportgen.dom.Text` object with the text specified by the `text` string.

`paragraphObj = Paragraph(text,styleName)` creates a paragraph having that specified style. The style specified by `styleName` must be defined in the template used for the document element to which this paragraph is appended.

`paragraphObj = Paragraph(docElementObj)` creates a paragraph containing the document element specified by `docElementObj` (for example, an image).

Input Arguments

`text` — Paragraph text

string

Paragraph text, specified as a string.

`styleName` — Style for the paragraph

string

The name of a style, specified as a string. 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 the new paragraph

a DOM object

A DOM object to include in a paragraph. You can specify these DOM objects:

- `mlreportgen.dom.ExternalLink`
- `mlreportgen.dom.Image`
- `mlreportgen.dom.InternalLink`
- `mlreportgen.dom.Text`
- `mlreportgen.dom.LinkTarget`

Output Arguments

`paragraphObj` — Paragraph

`mlreportgen.dom.Paragraph` object

Paragraph, represented by an `mlreportgen.dom.Paragraph` object.

Properties

`BackgroundColor` — Background color

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`Bold` — Option to use bold for text

[ ] (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 text 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

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`Children` — Children of this paragraph

array of `mlreportgen.dom.Element` objects

This read-only property lists children elements, such as an image (`mlreportgen.dom.Image`) object, that the paragraph contains.

`CustomAttributes` — Custom attributes of document element

array of `mlreportgen.dom.CustomAttribute` objects

The output format must support the custom attributes of this document element.

`FirstLineIndent` — Indentation amount for first line of paragraph

string

Amount by which to indent the first line of this paragraph relative to succeeding lines.

To create a hanging indent, in which all the lines are indented except for the first line, use a negative number.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the indentation is expressed. Use one of these abbreviations for the units for indentation.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`FontFamilyName` — Name of font family for paragraph text

string

The name of a font family.

To specify substitutions for this font, do not set this property. Instead create and add a `mlreportgen.dom.FontFamily` object to the `Style` property of this document element.

Setting the `FontFamilyName` property adds a corresponding `mlreportGen.dom.FontFamily` format object to the `Style` property for this document element. Removing the `FontFamilyName` property setting removes the object.

`FontSize` — Font size for paragraph text

string

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.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units. Use one of these abbreviations for the units:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`HAlign` — Horizontal alignment of this paragraph

string

Horizontal alignment for a paragraph, relative to page margins or table cell borders, specified as a string.

String	Description	Supported Output Types
`center`	Center the paragraph	All output types
`distribute`	Distribute all characters equally	Word
`justify`	Align left side of paragraph on left side of page, and right side of paragraph on the right side of the page	HTML and PDF

String	Description	Supported Output Types
`KashidaHigh`	Use widest Kashida length	Word
`KashidaLow`	Use lowest Kashida length	Word
`KashidaMedium`	Use medium Kashida length	Word
`left`	Align paragraph left	All output types
`right`	Align paragraph right	All output types
`ThaiDistribute`	Thai language justification	Word

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Italic` — Option to use italics for text

[ ] (default) | logical value

To use italics for text, 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 text 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`

`OuterLeftMargin` — Left indentation for paragraph

string

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.

To indent a paragraph from both the left and right margin of a page, do not set this property. Instead, add to the `Style` property of this paragraph a `mlreportgen.dom.OuterMargin` object specifying the left and right indentations.

Setting the `OuterLeftMargin` property adds a corresponding `mlreportGen.dom.OuterMargin` format object to the `Style` property for this document element. Removing the `OuterLeftMargin` property setting removes the object.

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the indentation is expressed. Use one of these abbreviations for the units for indentation.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`OutlineLevel` — Outline level of this paragraph

[ ] (default) | numeric value

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 table of contents. For example, to make a paragraph appear as a `Heading 1` (Word) or `h1` (HTML), set `OutlineLevel` to `1`.

Data Types: `int32`

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Strike` — Text strikethrough

string

The default for this property is []. You can set it to one of these values:

- `none` — Do not use strikethrough.
- `single` — Use a single line for strikethrough.
- `double` — Use a double line for strikethrough for Word documents.

Setting the `Strike` property adds a corresponding `mlreportGen.dom.Strike` format object to the `Style` property for this document element. Removing the `Strike` property setting removes the object.

`Style` — Paragraph formatting

cell array of format objects

A cell array of DOM format objects that specifies the formats for this paragraph style.

`StyleName` — Paragraph style name

string

The style specified by `StyleName` must be defined in the template used to create the document element to which this paragraph is appended.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Underline` — Type of underline, if any, for text

`[ ]` (default) | string

You can specify one of the following types of underlines.

Border String	Description	Supported Output Types
`dash`	Dashed underline	Word
`dashedHeavy`	Line with heavy dashes	Word
`dashLong`	Line with long dashes	Word
`dashLongHeavy`	Line with heavy long dashes	Word
`dashDotDotHeavy`	Line with heavy dashes with two dots between the dashes	Word
`dashDotHeavy`	Heavy dash-dot line	Word
`dotted`	Dotted line	Word
`dottedHeavy`	Thick dotted line	Word

Border String	Description	Supported Output Types
`dotDash`	Dot-dash line	Word
`dotDotDash`	Dot-dot-dash line	Word
`dashDotHeavy`	Heavy dot-dash line	Word
`double`	Double line	Word
`none`	Do not use underlining	All output types
`single`	Single line	All output types
`thick`	Thick line	Word
`wave`	Wavy line	Word
`waveyDouble`	Double wavy line	Word
`waveyHeavy`	Heavy wavy	Word
`words`	Underline non-space characters only	Word

If this property is empty and `StyleName` property of this document element specifies a style sheet style, the type of underline is determined by that style.

To specify the color as well as the type of the underline, do not set the `Underline` property. Instead, set the `Style` property of this document element to include an `mlreportgen.dom.Underline` format object that specifies the desired underline type and color.

Setting the `Underline` property adds a corresponding `mlreportgen.dom.Underline` format object to the `Style` property for this document element. Removing the `Underline` property setting removes the object.

`WhiteSpace` — White space and line breaks in text

[ ] (default) | string

To specify how to handle white space, use one of these values.

String	Description	Supported Output Types
`normal` (default)	For HTML and PDF, removes spaces at beginning and end of text string. Multiple spaces within	All output types

String	Description	Supported Output Types
	string collapse to single space. For Word, removes spaces at beginning and end of text string.	
`nowrap`	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
`preserve`	Preserves spaces and line feeds. Acts like the `<pre>` tag in HTML.	All output types
`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.	HTML and PDF
`pre-wrap`	Preserves white space. Text wraps when necessary and on line breaks	HTML and PDF

Setting the `WhiteSpace` property adds a corresponding `WhiteSpace` format object to `Style` property. Removing the `WhiteSpace` property setting removes the `WhiteSpace` object.

Methods

Method	Purpose
append	Append text, images, links, link targets, or custom elements to paragraph.
clone	Copy paragraph.

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('http://www.mathworks.com/', 'MathWorks');
p2 = Paragraph(link);
p2.BackgroundColor = 'yellow';
append(doc,p1);
append(doc,p2);

close(doc);
rptview('mydoc','html');
```

- “Add Content to a Report” on page 13-11

See Also

mlreportgen.dom.LineSpacing | mlreportgen.dom.Text | mlreportgen.dom.WhiteSpace

More About

- “Report Formatting Approaches” on page 13-21

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

`pdfPageFooterObj = PDFPageFooter()` creates an empty page footer based on the default PDF template.

`pdfPageFooterObj = PDFPageFooter(pageType)` creates an empty page footer for the specified type of page—odd, even, and first.

`pdfPageFooterObj = PDFPageFooter(pageType,templatePath)` creates an empty page footer based on the specified template.

`pdfPageFooterObj = PDFPageFooter(pageType,templatePath, docPartTemplateName)` creates an empty page footer for the specified type of page, based on the specified document part template in the specified template.

`pdfPageFooterObj = PDFPageFooter(pageType,templateSrc, docPartTemplateName)` creates an empty 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` — If there are no page footers defined with `pageType` set to `first`, the footer appears on the odd pages of a section and on the first page.
- `first` — Footer appears only on the first page of a section.
- `even` — Footer appears on the even pages of a section.

For example, to make different footers appear on the first page and on even pages, define two footers. Set `pageType` to `first` for one and to `even` for the other.

`templatePath` — Full path of footer template
string

Full path of footer template, specified as a string.

`docPartTemplateName` — Document part template name
string

Name of this part's template if it is stored in a template specified by the `templatePath` or `templateSrc` argument, specified as a string.

`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 a `mlreportgen.dom.DocumentPart` object for a document part.

Output Arguments

`pdfPageFooterObj` — 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 this object
cell array of objects

This read-only property lists child elements of this object.

`CurrentPageLayout` — Page footers ignore this property

`CurrentHoleId` — ID of current hole in document
string

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 document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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`— If there are no page footers defined with `pageType` set to `first`, footer appears on odd pages in a section and the first page.
- `first`— Appears only on the first page of a section.
- `even`— Appears on even pages in a section.

To have a footer appear on the first page and on even pages, define two footers, one with `pageType` set to `first` and the other with `pageType` set to `even`.

`Parent` — Parent of document element
DOM object

This read-only property lists the parent of this document element.

`Tag` — Tag for document element
string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`TemplatePath` — Path to template used for footer

string

Full path to the template to use for this footer, specified as a string.

Methods

Use `DocumentPageFooter` methods like you use the corresponding `Document` methods.

Method	Purpose
append	Append one of these DOM objects to the footer: • `CustomElement` • `FormalTable` • `Group` • `ExternalLink` • `Image` • `InternalLink` • `OrderedList` • `Paragraph` • `RawText` • `Table` • `Text` • `UnorderedList`
close	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);
```

- “Create Page Footers and Headers” on page 13-152

See Also

mlreportgen.dom.DOCXPageFooter | mlreportgen.dom.DOCXPageHeader | mlreportgen.dom.DOCXPageLayout | mlreportgen.dom.PDFPageHeader | mlreportgen.dom.PDFPageLayout

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

`pdfPageHeaderObj = PDFPageHeader()` creates an empty page header based on the default PDF template.

`pdfPageHeaderObj = PDFPageHeader(pageType,templatePath)` creates an empty page header for the specified type of page based on the specified template.

`pdfPageHeaderObj = PDFPageHeader(pageType,templatePath, docPartTemplateName)` creates an empty page header for the specified type of page, based on the specified document part template in the specified template.

`pdfPageHeaderObj = PDFPageHeader(pageType,templateSrc, docPartTemplateName)` creates an empty 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 appears on odd pages in a section and the first page, if there are no page headers defined with `pageType` set to `first`.
- `first`— Header appears only on the first page of a section.
- `even`— Header appears on even pages in a section.

For example, to have a header appear on the first page and on even pages, define two headers, one with `pageType` set to `first` and the other with `pageType` set to `even`.

`templatePath` — Full path of header template
string

Full path of footer template, specified as a string.

`docPartTemplateName` — Document part template name
string

Name of this part's template if it is stored in a template specified by the `templatePath` or `templateSrc` argument, specified as a string.

`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 a `mlreportgen.dom.DocumentPart` object for a document part.

Output Arguments

`pdfPageHeaderObj` — Page header for PDF document
`mlreportgen.dom.PDFPageHeader` object

Page header for a PDF document, returned as an `mlreportgen.dom.PDFPageHeader` object.

Properties

`Children` — Children of this object
cell array of objects

This read-only property lists child elements of this object.

`CurrentHoleId` — ID of current hole in document
string

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 [].

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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 appears on odd pages in a section and the first page, if there are no page headers defined with `pageType` set to `first`.
- `first`— Header appears only on the first page of a section.
- `even`— Header appears on even pages in a section.

For example, to have a header appear on the first page and on even pages, define two headers, one with `pageType` set to `first` and the other with `pageType` set to `even`.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`TemplatePath` — Full path to template for header

string

Full path to template to use for this header, specified as a string.

Methods

Use `PDFPageHeader` methods like you use the corresponding `Document` methods.

Method	Purpose
append	Append one of these DOM objects to the header: • `CustomElement` • `PDFPageLayout` • `FormalTable` • `Group` • `ExternalLink` • `Image` • `InternalLink` • `OrderedList` • `Paragraph` • `RawText` • `Table`

Method	Purpose
	• `Text` • `UnorderedList`
close	Close header.
fill	Fill template hole.
moveToNextHole	Move to next template hole.
open	Open header.

See Also

mlreportgen.dom.DOCXPageFooter | mlreportgen.dom.DOCXPageHeader | mlreportgen.dom.DOCXPageLayout | mlreportgen.dom.PDFPageFooter | mlreportgen.dom.PDFPageLayout

Related Examples

- “Create Page Footers and Headers” on page 13-152

Introduced in R2016a

mlreportgen.dom.PDFPageLayout class

Package: mlreportgen.dom

Page format and layout for PDF document section

Description

Use an `mlreportgen.dom.PDFPageLayout` object to define the page format, headers, and footers of a PDF document section.

Construction

`PageLayoutObj = PDFPageLayout()` creates an empty document page layout object.

Output Arguments

`PageLayoutObj` — Page layout object
`mlreportgen.dom.PDFPageLayout` object

Page format and layout for a PDF document section, returned as an `mlreportgen.dom.PDFPageLayout` object.

Properties

`Children` — Children of this object
cell array of objects

This read-only property lists child elements of this object.

`CustomAttributes` — Custom attributes of document element
array of `mlreportgen.dom.CustomAttribute` objects

The output format must support the custom attributes of this document element.

`FirstPageNumber` — Number of first page in section
integer

Number of the first page in a section, specified as an integer.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`PageFooters` — Page footers for this layout

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

`PageHeaders` — Page headers for this layout

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

`PageMargins` — Margin sizes for this layout

`mlreportgen.dom.PageMargins` object

Margin sizes for this layout, specified as an `mlreportgen.dom.PageMargins` object.

`PageNumberFormat` — Type of page numbering to use

`'a' | 'A' | 'i' | 'I' | 'n' | 'N'`

Type of page numbering to use, specified as one of these values:

- `'a'` — Lowercase alphabetic.
- `'A'` — Uppercase alphabetic.
- `'i'` — Lowercase roman numerals.
- `'I'` — Uppercase roman numerals.

- `'n'` or `'N'` — Numeric page numbers.

`PageSize` — Size and orientation of pages in this layout

`mlreportgen.dom.PageSize` object

Size and orientation of pages in this layout, specified as an `mlreportgen.dom.PageSize` object.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`SectionBreak` — Section break options

`'Next Page' | 'Odd Page' | 'Even Page'`

Option to create a section break for this layout, specified as one of these values:

- `'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.

`Style` — Formats to apply to layout

array of format objects

Formats to apply to this layout, specified as an array of format objects. The formats you specify using this property override the same formats defined by the style applied with the `StyleName` property. Formats that do not apply to a page layout are ignored.

`StyleName` — Ignored by page layouts

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Examples

Change Page Margins of a Document Section

Create a PDF report using the default template. Open the document and assign the `CurrentPageLayout` property to a variable. Change the left and right margins for this layout.

```
import mlreportgen.dom.*;
d = Document('mydoc','pdf');

open(d);
s = d.CurrentPageLayout;
s.PageMargins.Left = '2in';
s.PageMargins.Right = '2in';
p = Paragraph('Hello World');
append(d,p);

close(d);
rptview(d.OutputPath);
```

See Also

mlreportgen.dom.DocumentPart | mlreportgen.dom.PageMargins | mlreportgen.dom.PageRawFormat | mlreportgen.dom.PageSize | mlreportgen.dom.PDFPageFooter | mlreportgen.dom.PDFPageHeader

More About

- “Report Formatting Approaches” on page 13-21

Introduced in R2016a

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
string

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 document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Source` — Source DOM object message originates from
a DOM object

Source DOM object from which the message originates.

`Tag` — Tag for document element
string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Text` — Text of message
string

Message text, specified as a string.

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);
dispatch(dispatcher,ProgressMessage('starting chapter',d));
```

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);
```

Run report and delete the listener.

```
close(d);
rptview('test','html');
```

Delete the listener to avoid duplicate reporting of message objects during a MATLAB session.

```
delete(l);
```

- “Display Report Generation Messages” on page 13-119

See Also

mlreportgen.dom.MessageDispatcher.dispatch

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

string

HTML markup, specified as a string. 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

string

Word XML markup or HTML markup, specified as a string. 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 string.

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

string

Word XML markup, specified as a string. 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

string

HTML markup, specified as a string. 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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

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');
```

- “Add Content to a Report” on page 13-11

See Also

mlreportgen.dom.CustomAttribute

mlreportgen.dom.RepeatAsHeaderRow class

Package: mlreportgen.dom

Repeat table row

Description

Specifies to repeat a table row on each new page when a table flows across multiple pages. This format applies only to Microsoft Word documents.

Construction

`repeatAsHeaderRowObj = RepeatAsHeaderRow()` repeats table row on each new page when a table flows across multiple pages.

`repeatAsHeaderRowObj = RepeatAsHeaderRow(onOff)` repeats table row on each new page if `onOff` is `true`.

Input Arguments

`onOff` — Controls table row repeating on each new page
`true` (default)

Specify one of the following logical values:

- `true` or `1` — Table row repeats on each new page when a table flows across multiple pages.
- `false` or `0` — Table row does not repeat on each new page when a table flows across multiple pages.

Data Types: `logical`

Output Arguments

`repeatAsHeaderRowObj` — Repeat table row
`mlreportgen.dom.RepeatAsHeaderRow` object

Repeat table row, represented by an `mlreportgen.dom.RepeatAsHeaderRow` object.

Properties

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Repeat table row on each new page

`true` (default)

Possible logical values are:

- `true` or `1` — Table row repeats on each new page when a table flows across multiple pages.
- `false` or `0` — Table row does not repeat on each new page when a table flows across multiple pages.

Data Types: `logical`

Examples

Repeat Table Row

Create a table with nor repeating row.

```
import mlreportgen.dom.*;
doctype = 'docx';
d = Document('repeatHeader',doctype);

append(d,'Table 1');
table = Table(ones(15, 2));
table.Style = {Border('solid'),RowSep('solid')};
append(d,table);
```

Create a second table with repeated row with a table row that cannot break across pages.

```
append(d,'Table 2');
table = Table(ones(15,2));
table.entry(1,1).Children(1).Content = 'Header A';
table.entry(1,2).Children(1).Content = 'Header B';
table.row(1).Style = {RepeatAsHeaderRow(true)};
table.Style = {Border('solid'),RowSep('solid')};
append(d, table);
table.row(6).Style = {AllowBreakAcrossPages(false)};
table.entry(6,1).Children(1).Content = ...
     'Start this row on new page if it does not fit on current page';
for i=2:10
     table.entry(6,1).append(Paragraph(Text(i)));
end

close(d);
rptview(d.OutputPath,doctype);
```

Generate the report.

```
close(d);
rptview(d.OutputPath,doctype);
```

- “Create and Format Tables” on page 13-69

See Also

mlreportgen.dom.AllowBreakAcrossPages | mlreportgen.dom.TableHeader

More About

- “Report Formatting Approaches” on page 13-21

mlreportgen.dom.ResizeToFitContents class

Package: mlreportgen.dom

Allow table to resize its columns

Description

Specifies whether a table resizes its columns to fit content.

Construction

`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`

Output Arguments

`resizeToFitContentsObj` — Allow table to resize its columns

`mlreportgen.dom.ResizeToFitContents` object

Specification of whether a table resizes its columns to fit content or wraps content, represented by an `mlreportgen.dom.ResizeToFitContents` object.

Properties

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`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`

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(ones(4,4));
```

```
table1.entry(1,2).Children(1).Content = 'MathWorks';

table1.Style = {ResizeToFitContents(true),Width('1in'), ...
    Border('solid'),RowSep('solid'),ColSep('solid')};

table1.TableEntriesStyle = {Width('0.25in')};
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(ones(4, 4));
table2.entry(1,2).Children(1).Content = 'MathWorks';

table2.Style = {ResizeToFitContents(false),Width('1in'), ...
    Border('solid'), RowSep('solid'),ColSep('solid')};

table2.TableEntriesStyle = {Width('0.25in')};
append(d,table2);
```

Run the report.

```
close(d);
rptview(d.OutputPath,doctype);
```

See Also

mlreportgen.dom.FormalTable | mlreportgen.dom.Table | mlreportgen.dom.TableColSpec | mlreportgen.dom.TableColSpecGroup

More About

- “Report Formatting Approaches” on page 13-21

mlreportgen.dom.RowHeight class

Package: mlreportgen.dom

Height of table row

Description

Specifies the height of a table row.

Construction

`rowHeightObj = RowHeight()` specifies row height to be 1 inch.

`rowHeightObj = RowHeight(height)` sets a row to the specified height.

`rowHeightObj = RowHeight(height,heightType)` sets a row to be *exactly* the specified height or *at least* the specified height. Applies only to Microsoft Word documents.

Input Arguments

`height` — Height of table row

`'1in'` (default)

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the height is expressed. The following abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`heightType` — Use specified height or height that is at least specified height

`'exact' | 'atleast'`

String that defines a row to be exactly the specified height or to be the specified height or taller.

Output Arguments

`rowHeightObj` — Height of table row

`mlreportgen.dom.RowHeight` object

Row height, represented by an `mlreportgen.dom.RowHeight` object.

Properties

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Type` — Use specified height or height that is at least specified height

`'exact' | 'atleast'`

String that defines a row to be exactly the specified height or to be the specified height or taller.

`Value` — Height of table row

`'1in'` (default)

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the height is expressed. The following abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

Examples

Specify Table Row Heights

Create a table with two rows. The first row has a variable 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 = '1in';
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');
```

- “Create and Format Tables” on page 13-69

See Also

mlreportgen.dom.TableRow

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

string

Line style of the table row separator, specified as a string.

String	Applies To	
	Word	HTML and PDF
`'dashed'`	✓	✓
`'dashdotstroked'`	✓	
`'dashsmallgap'`	✓	
`'dotted'`	✓	✓

String	Applies To	
	Word	HTML and PDF
'dotdash'	✓	
'dotdotdash'	✓	
'double'	✓	✓
'doublewave'	✓	
'inset'	✓	✓
'none'	✓	✓
'outset'	✓	✓
'single'	✓	
'solid'		✓
'thick'	✓	
'thickthinlargegap'	✓	
'thickthinmediumgap'	✓	
'thickthinsmallgap'	✓	
'thinthicklargegap'	✓	
'thinthickmediumgap'	✓	
'thinthicksmallgap'	✓	
'thinthickthinlargegap'	✓	
'thinthickthinmediumgap'	✓	

String	Applies To	
	Word	HTML and PDF
`'thinthickthinsmallgap'`	✓	
`'threedemboss'`	✓	
`'threedengrave'`	✓	
`'triple'`	✓	
`'wave'`	✓	

`color` — Color of table row separator
string

String specifying the color of the table row separator. String can be a color, such as `'red'` or a hexadecimal RGB value, such as `'#0000ff'`.

`width` — Width of table row separator
string

String specifying the color of the table row separator. String 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
- `pi` — picas
- `pt` — points
- `px` — pixels

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
string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`Id` — ID for document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Style` — Line style of table row separator
string

Line style for the row separator. See the description of the `style` input argument for a list of possible values.

`Tag` — Tag for document element
string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Width` — Table row separator width
string

String specifying the width of the table row separator. String 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
- `pi` — picas
- `pt` — points
- `px` — pixels

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);
```

- “Create and Format Tables” on page 13-69

See Also

mlreportgen.dom.Border | mlreportgen.dom.ColSep | mlreportgen.dom.TableEntry | mlreportgen.dom.TableRow

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

Construction

`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

[ ] (default) | logical value

A setting of `false` (or `0`) does not scale the image. A setting of `true` (or `1`) scales the image to fit between the margins of the page or within a table entry.

Data Types: `logical`

Output Arguments

`scaleToFitObj` — Scale image to fit page

`mlreportgen.dom.ScaleToFit` object

Scale image to fit page, returned as an `mlreportgen.dom.ScaleToFit` object.

Properties

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Scale image to fit page

`[ ]` (default) | logical value

The possible values are:

- `false` or `0` — Do not scale image.
- `true` or `1`— Scale image to fit between the margins or in table entry.

Data Types: `logical`

Examples

Scale Image to Fit Margins andTable 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 ScaletoFit 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(''1in''), Width(''1in'')}');
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('1in'), Width('1in')};
table.entry(1,2).Border = 'solid';
append(d,table);

close(d);
```

```
rptview(d.OutputPath);
```

- “Create and Format Images” on page 13-86

See Also

mlreportgen.dom.Height | mlreportgen.dom.Image | mlreportgen.dom.Width

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` — Specifies strike type

string

String specifying the strike type. Choices are:

- `'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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Specifies strike type

string

String specifying the strike type. Choices are:

- `'single'` — Single horizontal line (default)
- `'none'` — No strikethrough line
- `'double'` — Double horizontal line

See Also

mlreportgen.dom.Underline

More About

- “Report Formatting Approaches” on page 13-21

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

string

Name of style of object to reference, specified as a string.

Output Arguments

`styleRef` — Referenced object

`mlreportgen.dom.StyleRef` object

Referenced object, returned as an `mlreportgen.dom.StyleRef` object.

Properties

`Children` — Children of this object

cell array of objects

This read-only property lists child elements of this object.

`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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`RefOutlineLevel` — Outline (heading) level of object to reference

positive integer

`RefStyleName` — Stylesheet style name of object to reference

string

Stylesheet style to apply to the reference, specified as a string.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`Style` — Format specification

array of format objects

Format objects that specify the format of a document element.

`StyleName` — Style to apply from stylesheet

string

Name of the style to apply from the stylesheet, specified as a string.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, 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 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.DOCXPageFooter | mlreportgen.dom.DOCXPageHeader | mlreportgen.dom.Heading | mlreportgen.dom.Heading1 | mlreportgen.dom.PageRef | mlreportgen.dom.Paragraph | mlreportgen.dom.PDFPageFooter | mlreportgen.dom.PDFPageHeader

More About

- “Report Formatting Approaches” on page 13-21

Introduced in R2016a

mlreportgen.dom.Table class

Package: mlreportgen.dom

Create table

Description

Use an `mlreportgen.dom.Table` object to define a table. Append rows and table entries to add content to the table. You can define column properties.

Construction

`tableObj = Table(nCols)` creates an empty table having the specified number of columns. Use this constructor as the starting point for creating a table from scratch.

`tableObj = Table(array)` returns a table whose content is specified by a two-dimensional numeric array or a two-dimensional cell array of MATLAB data types and DOM objects. The constructor converts basic MATLAB types to corresponding DOM types, e.g., strings to Text objects.

`tableObj = Table(array,'StyleName')` creates a table having the specified style. The style specified by `StyleName` must be defined in the template used to create the document to which this table is appended.

Input Arguments

`nCols` — Number of table columns

double

Number of table columns, specified as a double.

Data Types: `double`

`array` — Array of values to include in a table

two-dimensional numeric array | two-dimensional cell array of MATLAB or DOM objects

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`

`styleName` — Style for table
string

The style specified by `styleName` must be defined in the template of the document to which this table is appended.

Output Arguments

`tableObj` — Table
`mlreportgen.dom.Table` object

Table, represented by an `mlreportgen.dom.Table` object.

Properties

`BackgroundColor` — Background color
string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`Border` — Type of border to draw
string

Specify one of the following as a string.

Border String	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
`hidden`	No line See discussion below this table.	HTML and PDF
`inset`	3-D effect line	All output types
`none`	No line See discussion below this table.	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 String	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 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)

string

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

string

You can specify:

- Name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`BorderWidth` — Table border width

string

String specifying the width of the border. The string 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`Children` — Children of this object

cell array of objects

This read-only property lists child elements of this object.

`ColSep` — Style of line separating columns

string

The style the line separating the columns of a table or table section (header, body, footer), as specified by a `mlreportgen.dom.ColSep` object.

See the description of the `Border` property for a description of the possible values.

`ColSepColor` — Color of line separating columns

string

You can specify:

- Name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`ColSepWidth` — Width of line separating table columns

string

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the width is expressed. Use one of these abbreviations for the units of a width.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

For example, for a three pica wide column separator, set the `ColSepWidth` property to `3pi`.

`ColSpecGroups` — Properties of group of columns in table

array of `mlreportgen.dom.TableColSpecGroups` objects

An array of `mlreportgen.dom.TableColSpecGroups` 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, etc. Specify the number of columns belonging to each group using the `Span` property of the `TableColSpecGroups` 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, etc.

`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` — Text flow direction

string

String specifying the direction for text to flow.

- `'ltr'` — flow from left to right
- `'rtl'` — flow from right to left

`HAlign` — Horizontal alignment of this table

string

Possible values are:

- `center`
- `left`
- `right`

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`OuterLeftMargin` — Left margin (indentation) of document element

string

String specifying the left indentation. The string 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`RowSep` — Style of lines separating rows

string

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

string

String specifying 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

string

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the width is expressed. Use one of these abbreviations for the units of a width.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`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 stylesheet style specified by the `StyleName` property.

`StyleName` — Style in document or document part stylesheet

string

Name of a style specified in the stylesheet 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.

`TableEntriesHAlign` — Horizontal alignment of table entries

`center` (default) | `left` | `right`

Data Types: `char`

`TableEntriesVAlign` — Vertical alignment of table cell content

string

Possible values are:

- `top`
- `middle`
- `bottom`

`TableEntriesInnerMargin` — Inner margin for table entries

string

The inner margin is the margin between table cell content and the cell borders.

The string 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`TableEntriesStyle` — Style to use for table entries

cell array

Cell array of format objects that specify the format for table entries.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Width` — Table width

string

String representing 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`, where `Units` is an abbreviation for the units in which the width is expressed.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

Methods

Method	Purpose
`append`	Append a content to a table.
`clone`	Clone this table.
`entry`	Get a table entry.

Method	Purpose
row	Create a table row.

Examples

Create a Table

```
import mlreportgen.dom.*;
d = Document('myreport', 'html');
open(d);
t = Table(magic(5));
t.Style = {RowHeight('1in')};
t.Border = 'solid';
t.BorderWidth = '2pt';
t.ColSep = 'solid';
t.ColSepWidth = '1';
t.RowSep = 'solid';
t.RowSepWidth = '1';
```

- “Create and Format Tables” on page 13-69

See Also

mlreportgen.dom.FormalTable | mlreportgen.dom.TableBody | mlreportgen.dom.TableEntry | mlreportgen.dom.TableFooter | mlreportgen.dom.TableHeader | mlreportgen.dom.TableHeaderEntry | mlreportgen.dom.TableRow

mlreportgen.dom.TableBody class

Package: mlreportgen.dom

Body of formal table

Description

Specifies the body of a formal table

Properties

`Children` — Children of this object

cell array of objects

This read-only property lists child elements of this object.

`ColSep` — Style of line separating columns

string

The style the line separating the columns of a table or table section (header, body, footer), as specified by a `mlreportgen.dom.ColSep` object.

See the description of the `Border` property for a description of the possible values.

`ColSepColor` — Color of line separating columns

string

You can specify:

- Name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`ColSepWidth` — Width of line separating table columns

string

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the width is expressed. Use one of these abbreviations for the units of a width.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

For example, for a three pica wide column separator, set the `ColSepWidth` property to `3pi`.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`RowSep` — Style of lines separating rows

string

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

string

String specifying 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

This read-only property lists the parent of this document element.

`Style` — Format specification

array of format objects

Format objects that specify the format of a document element.

`Stylename` — Table body style name

string

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`

Data Types: `char`

`TableEntriesVAlign` — Vertical alignment of table cell content

string

Possible values are:

- `top`
- `middle`
- `bottom`

`TableEntriesInnerMargin` — Inner margin for table entries

string

The inner margin is the margin between table cell content and the cell borders.

The string 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
- `pi` — picas
- `pt` — points

- `px` — pixels

`TableEntriesStyle` — Style to use for table entries

cell array

Cell array of format objects that specify the format for table entries.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Methods

Method	Purpose
append	Appends content to a table body.
`entry`	Get a table entry.
`row`	Create a table row.

See Also

mlreportgen.dom.FormalTable | mlreportgen.dom.Table | mlreportgen.dom.TableEntry | mlreportgen.dom.TableFooter | mlreportgen.dom.TableHeader | mlreportgen.dom.TableHeaderEntry | mlreportgen.dom.TableRow

Related Examples

- “Create and Format Tables” on page 13-69

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

The output format must support the custom attributes of this document element.

`Id` — ID for document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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 document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

See Also

mlreportgen.dom.FormalTable | mlreportgen.dom.ResizeToFitContents | mlreportgen.dom.Table | mlreportgen.dom.TableColSpecGroup

Related Examples

- “Create and Format Tables” on page 13-69

mlreportgen.dom.TableColSpecGroup class

Package: mlreportgen.dom

Define style for group of table columns

Description

Define the style for a group of table columns. Use a `TableColSpec` object to override formats specified by a `TableColSpecGroup` object.

Construction

`colSpecGroupObj = TableColSpecGroup()` creates a column specification that spans an entire table.

Output Arguments

`colSpecGroupObj` — Table column specification
`mlreportgen.dom.TableColSpecGroup` object

Specification of formats for a group of table columns, represented by an `mlreportgen.dom.TableColSpecGroup` object.

Properties

`ColSpec` — Type of line to draw between columns of this table
this property accepts the same set of values as the `Border` property

Data Types: `char`

`CustomAttributes` — Custom attributes of document element
array of `mlreportgen.dom.CustomAttribute` objects

The output format must support the custom attributes of this document element.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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` — Format specification

array of format objects

Format objects that specify the format of a document element.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Examples

Make the First Column Green and Remaining Columns Red

```
import mlreportgen.dom.*
d = Document('mydoc','docx');
append(d,'Table');

grps(1) = TableColSpecGroup;
grps(1).Style = {Color('red')};
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.OutputPath);
```

- “Create and Format Tables” on page 13-69

See Also

mlreportgen.dom.FormalTable | mlreportgen.dom.ResizeToFitContents | mlreportgen.dom.Table | mlreportgen.dom.TableColSpec

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.

Construction

`entryObj = TableEntry()` creates an empty table entry.

`entryObj = TableEntry(text)` creates a table entry using the text included in the specified `mlreportgen.dom.Text` object.

`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 a `mlreportgen.dom.Paragraph` object.

Input Arguments

`text` — Table entry text
string

String containing the text for the table entry.

`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
string

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`

Output Arguments

`entryObj` — Table entry
`mlreportgen.dom.TableEntry` object

Table entry, returned as an `mlreportgen.dom.TableEntry` object

Properties

`Border` — Type of border to draw
string

Specify one of the following as a string.

Border String	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
`hidden`	No line See discussion below this table.	HTML and PDF
`inset`	3-D effect line	All output types
`none`	No line See discussion below this table.	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 String	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 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

`BorderColor` — Border color

string

You can specify:

- Name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`BorderWidth` — Table border width

string

String specifying the width of the border. The string 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`InnerMargin` — Inner margin (padding) around entry

string

String specifying the inner margin. String 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`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 stylesheet style specified by the `StyleName` property.

`StyleName` — Style in document or document part stylesheet

string

Name of a style specified in the stylesheet 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 document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`VAlign` — Vertical alignment table cell content
string

Possible values are:

- `top`
- `bottom`
- `middle`

Methods

Use `TableEntry.append` and `TableEntry.clone` methods the same way you use `Paragraph.append` and `Paragraph.clone`.

Method	Purpose
append	Append text, paragraphs, images, tables, and other elements to this table entry.
`clone`	Clone this table entry.

See Also

mlreportgen.dom.FormalTable | mlreportgen.dom.Table | mlreportgen.dom.TableBody | mlreportgen.dom.TableFooter | mlreportgen.dom.TableHeader | mlreportgen.dom.TableHeaderEntry | mlreportgen.dom.TableHeaderEntry | mlreportgen.dom.TableRow | mlreportgen.dom.TextOrientation

Related Examples

- “Create and Format Tables” on page 13-69

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 this object

cell array of objects

This read-only property lists child elements of this object.

`ColSep` — Style of line separating columns

string

The style the line separating the columns of a table or table section (header, body, footer), as specified by a `mlreportgen.dom.ColSep` object.

See the description of the `Border` property for a description of the possible values.

`ColSepColor` — Color of line separating columns

string

You can specify:

- Name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`ColSepWidth` — Width of line separating table columns

string

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the width is expressed. Use one of these abbreviations for the units of a width.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

For example, for a three pica wide column separator, set the `ColSepWidth` property to `3pi`.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`RowSep` — Style of lines separating rows

string

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

string

String specifying 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

string

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the width is expressed. Use one of these abbreviations for the units of a width.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`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 stylesheet style specified by the `StyleName` property.

`StyleName` — Style in the document or document part stylesheet

string

Name of a style specified in the stylesheet 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`

Data Types: `char`

`TableEntriesInnerMargin` — Inner margin for table entries

string

The inner margin is the margin between table cell content and the cell borders.

The string 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`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

string

Possible values are:

- `top`
- `middle`
- `bottom`

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Methods

Method	Purpose
`append`	Appends a row of table entries to this table footer.

Method	Purpose
`entry`	Get a footer entry
`row`	Get a footer row

See Also

mlreportgen.dom.FormalTable | mlreportgen.dom.TableBody | mlreportgen.dom.TableEntry | mlreportgen.dom.TableHeader | mlreportgen.dom.TableHeaderEntry | mlreportgen.dom.TableRow

Related Examples

- “Create and Format Tables” on page 13-69

mlreportgen.dom.TableHeader class

Package: mlreportgen.dom

Table header

Description

Table header for labeling columns.

Properties

`Children` — Children of this object

cell array of objects

This read-only property lists child elements of this object.

`ColSep` — Style of line separating columns

string

The style the line separating the columns of a table or table section (header, body, footer), as specified by a `mlreportgen.dom.ColSep` object.

See the description of the `Border` property for a description of the possible values.

`ColSepColor` — Color of line separating columns

string

You can specify:

- Name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`ColSepWidth` — Width of line separating table columns

string

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the width is expressed. Use one of these abbreviations for the units of a width.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

For example, for a three pica wide column separator, set the `ColSepWidth` property to `3pi`.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`RowSep` — Style of lines separating rows

string

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

string

String specifying 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

string

The string has the format `valueUnits`, where `Units` is an abbreviation for the units in which the width is expressed. Use one of these abbreviations for the units of a width.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`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 stylesheet style specified by the `StyleName` property.

`StyleName` — Style in the document or document part stylesheet

string

Name of a style specified in the stylesheet 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`

Data Types: `char`

`TableEntriesInnerMargin` — Inner margin for table entries

string

The inner margin is the margin between table cell content and the cell borders.

The string 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`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

string

Possible values are:

- `top`
- `middle`
- `bottom`

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Methods

Method	Purpose
append	Appends a row of table entries to this table header.
entry	Get a header entry.
row	Get a header row.

See Also

mlreportgen.dom.FormalTable | mlreportgen.dom.TableBody | mlreportgen.dom.TableEntry | mlreportgen.dom.TableFooter | mlreportgen.dom.TableHeaderEntry | mlreportgen.dom.TableRow

Related Examples

- “Create and Format Tables” on page 13-69

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

string

Specify one of the following as a string.

Border String	Description	Supported Output Types
`dashed`	Dashed line	All output types

Border String	Description	Supported 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 See discussion below this table.	HTML and PDF
`inset`	3-D effect line	All output types
`none`	No line See discussion below this table.	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

Border String	Description	Supported Output Types
`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

`BorderColor` — Border color

string

You can specify:

- Name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`BorderWidth` — Table border width

string

String specifying the width of the border. The string 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`InnerMargin` — Inner margin (padding) around entry

string

String specifying the inner margin. String 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`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 stylesheet style specified by the `StyleName` property.

`StyleName` — Style in document or document part stylesheet

string

Name of a style specified in the stylesheet 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 document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`VAlign` — Vertical alignment table cell content

string

Possible values are:

- `top`
- `bottom`
- `middle`

Methods

Use `TableHeaderEntry.append` and `TableHeaderEntry.clone` methods the same way you use `Paragraph.append` and `Paragraph.clone`.

Method	Purpose
append	Append text, paragraphs, images, tables, and other elements to this entry.
`clone`	Clone this table header entry.

See Also

mlreportgen.dom.FormalTable | mlreportgen.dom.TableBody | mlreportgen.dom.TableFooter | mlreportgen.dom.TableHeader | mlreportgen.dom.TableRow

Related Examples

- “Create and Format Tables” on page 13-69

mlreportgen.dom.TableRow class

Package: mlreportgen.dom

Table row

Description

Creates a table row.

Construction

`tableRowObj = TableRow()` creates an empty table row.

Output Arguments

`tableRowObj` — Table row
`mlreportgen.dom.TableRow` object

Table row, represented by an `mlreportgen.dom.TableRow` object.

Properties

`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.

`Entries` — Table entries in this row
array of `mlreportgen.dom.TableEntry` objects

Table entries in this row.

`Height` — Height of table row
string

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the height is expressed. The following abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

Alternatively, you can specify the table row height using the `TableRow.Style` property. For example:

```
TableRow.Style = {RowHeight('.5in')};
```

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Style` — Style of table row

array of format objects

Array of format objects (such as `Bold` objects) that specify the style of the table row.

`StyleName` — Style of table row

string

Name of a style specified in the stylesheet of the HTML document or document part containing this row. This property is ignored for Word output.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Methods

Method	Purpose
append	Append entries to this row.
clone	Clone this row.

See Also

mlreportgen.dom.FormalTable | mlreportgen.dom.Table | mlreportgen.dom.TableBody | mlreportgen.dom.TableEntry | mlreportgen.dom.TableFooter | mlreportgen.dom.TableHeader

Related Examples

- “Create and Format Tables” on page 13-69

mlreportgen.dom.Template class

Package: mlreportgen.dom

Create report template object for Word or HTML

Description

Create a Word or HTML template object.

Use `mlreportgen.dom.Template` objects to create templates programmatically. For example, you can append DOM content (e.g., `Text`, `Paragraph`, `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 programmatically, using the DOM API. Create a `Template` object and usemlreportgen.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.htmtx`.

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 12-470 for an example.

`templateObj = Template(templatePath,type,sourceTemplatePath)` creates a template based on the template `sourceTemplatePath`.

Input Arguments

`templatePath` — Template to create

string

Full path of template you want to create, specified as a string. You can use an extension to create a template of the corresponding type:

- `.htmtx` for HTML (default)
- `.docx` for Word
- `.htmx` for single-file HTML

`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

If you use an extension with the `templatePath` input argument, the `type` argument must match.

`sourceTemplatePath` — Basis for new template

string

Template that is the basis for the new template, specified as a string. 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 this object

cell array of objects

This read-only property lists child elements of this object.

`CurrentHoleId` — ID of current hole in document

string

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

string

Custom content for HTML header, specified as a string.

Data Types: `char`

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`OutputPath` — Path of output file or folder

string

Path of the output file or folder, specified as a string. 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'`

Packaging for output files generated, specified as one of these values:

- `zipped` (default)
- `unzipped`
- `both`

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. 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`.

The `unzipped` option applies only to Word (`docx`) and multifile HTML (`htmx`) output. 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`, for Word and multifile HTML output. 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) | string

String 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

string

The full path to the HTML or Word template to use, specified as a string.

`TitleBarText` — Title for HTML browser title bar

string

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'`

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.

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
append	Append document element to the document.
close	Close this document.
createTemplate	Create default template.
fill	Fill document hole.
getCoreProperties	Get core properties of document.
getImageDirectory	Get image directory for the document.
getImagePrefix	Get generated image name prefix for the document.
getMainPartPath	Get relative path of main part of output document.
getOPCMainPart	Get full path of main part of output document.
moveToNextHole	Move to next template hole.
open	Open this document.
package	Append file to OPC package of document.
setCoreProperties	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 string
p = Paragraph('Author: ');
p.StyleName = 'Subtitle';
p.Style = {OuterMargin('0','0','1in','1in')};
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)
```

- “Create a Microsoft Word Template” on page 13-124
- “Create an HTML or PDF Template” on page 13-135
- “Use Style Sheet Styles” on page 13-24

See Also

mlreportgen.dom.Document.createTemplate | mlreportgen.dom.TemplateHole | `rptview`

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

string

The ID for the template hole, specified by a string.

`description` — Description for template hole

string

Description for the template hole, specified as a string. 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

string

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

string

Description for the template hole, specified as a string. 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

string

ID of this template hole.

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Methods

Method	Purpose
clone Use `TemplateHole.clone` in a similar way to how you use `Paragraph.clone`.	Clone this hole object.

See Also

mlreportgen.dom.Document.createTemplate | mlreportgen.dom.Template | `rptview`

Related Examples

- “Create a Microsoft Word Template” on page 13-124
- “Create an HTML or PDF Template” on page 13-135
- “Use Style Sheet Styles” on page 13-24

mlreportgen.dom.Text class

Package: mlreportgen.dom

Text object

Description

Text string to include in a document element

Construction

`textObj = Text()` creates an empty text object.

`textObj = Text(text)` creates a text object containing the specified text string.

`textObj = Text(text,styleName)` creates a text object containing the specified text string using the specified style. The style must be defined in the style sheet of the template of the document to which this text object is appended.

Input Arguments

`text` — Text string

array of chars

Array of chars containing the text

Data Types: `char`

`styleName` — Style for the text

`mlreportgen.dom.StyleName` object

The style specified by `styleName` must be defined in the template used to create the document to which this text is appended.

Data Types: `char`

Output Arguments

`textObj` — Text string

`mlreportgen.dom.Text` object

Text string, represented by an `mlreportgen.dom.Text` object.

Properties

`BackgroundColor` — Background color

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`Bold` — Option to use bold for text

`[ ]` (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 text 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

string

Specify one of these as a string:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- A hexadecimal RGB (truecolor) value as `#RRGGBB`. For example, `#0000ff` is a shade of blue.

`Content` — Text string contained by this document element

string

Text string contained by this document element.

`CustomAttributes` — Custom attributes of document element

array of `mlreportgen.dom.CustomAttribute` objects

The output format must support the custom attributes of this document element.

`FontFamilyName` — Name of font family for text

string

The name of a font family.

To specify substitutions for this font, do not set this property. Instead create and add a `mlreportgen.dom.FontFamily` object to the `Style` property of this document element.

Setting the `FontFamilyName` property adds a corresponding `mlreportGen.dom.FontFamily` format object to the `Style` property for this document element. Removing the `FontFamilyName` property setting removes the object.

`FontSize` — Font size for text

string

If you need to specify substitutions for this font, do not set this property. Instead create and add a `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.

String having the format `valueUnits`, where `Units` is an abbreviation for the units in which the font size is expressed. Use one of these abbreviations for the units for the font size.

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters

- `pi` — picas
- `pt` — points
- `px` — pixels

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Italic` — Option to use italics for text

[ ] (default) | logical value

To use italics for text, 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 text 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

string

The default for this property is []. You can set it to one of these values:

- `none` — Do not use strikethrough.
- `single` — Use a single line for strikethrough.
- `double` — Use a double line for strikethrough for Word documents.

Setting the `Strike` property adds a corresponding `mlreportGen.dom.Strike` format object to the `Style` property for this document element. Removing the `Strike` property setting removes the object.

`Style` — Text formatting

array of DOM style objects

An array of DOM style objects that specifies the format for the text.

`StyleName` — Style for the text

string

The style specified by `StyleName` must be defined in the template used to create the document element to which this text is appended.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Underline` — Type of underline, if any, for text

[ ] (default) | string

You can specify one of the following types of underlines.

Border String	Description	Supported Output Types
`dash`	Dashed underline	Word
`dashedHeavy`	Line with heavy dashes	Word
`dashLong`	Line with long dashes	Word
`dashLongHeavy`	Line with heavy long dashes	Word
`dashDotDotHeavy`	Line with heavy dashes with two dots between the dashes	Word
`dashDotHeavy`	Heavy dash-dot line	Word
`dotted`	Dotted line	Word
`dottedHeavy`	Thick dotted line	Word
`dotDash`	Dot-dash line	Word
`dotDotDash`	Dot-dot-dash line	Word
`dashDotHeavy`	Heavy dot-dash line	Word
`double`	Double line	Word
`none`	Do not use underlining	All output types
`single`	Single line	All output types

Border String	Description	Supported Output Types
`thick`	Thick line	Word
`wave`	Wavy line	Word
`waveyDouble`	Double wavy line	Word
`waveyHeavy`	Heavy wavy	Word
`words`	Underline non-space characters only	Word

If this property is empty and `StyleName` property of this document element specifies a style sheet style, the type of underline is determined by that style.

To specify the color as well as the type of the underline, do not set the `Underline` property. Instead, set the `Style` property of this document element to include an `mlreportgen.dom.Underline` format object that specifies the desired underline type and color.

Setting the `Underline` property adds a corresponding `mlreportgen.dom.Underline` format object to the `Style` property for this document element. Removing the `Underline` property setting removes the object.

`WhiteSpace` — White space and line breaks in text

`[ ]` (default) | string

To specify how to handle white space, use one of these values. The `WhiteSpace` property does not apply to PDF format for `Text` objects.

String	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 only on line breaks. Acts like the `<pre>` tag in HTML.	Word and HTML See below for details.

String	Description	Supported Output Types
`'pre'`	Preserves white space. Text wraps only on line breaks. Acts like the `<pre>` tag 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 rather than 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.

Methods

Use the `Text.append` and `Text.clone` methods the same way you use the `Paragraph.append` and `Paragraph.clone` methods.

Method	Purpose
append	Append a custom element to this text object.
clone	Clone this text object

See Also

mlreportgen.dom.CharEntity | mlreportgen.dom.CustomText | mlreportgen.dom.Paragraph

Related Examples

- “Add Content to a Report” on page 13-11

More About

- "Report Formatting Approaches" on page 13-21

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 a string. Valid values are:

- `'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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Text orientation

`'horizontal' | 'down' | 'up'`

Text orientation, specified as a string. Valid values are:

- `'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

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);
```

- “Create and Format Tables” on page 13-69

See Also

mlreportgen.dom.TableEntry

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 this object
cell array of objects

This read-only property lists child elements of this object.

`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 document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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

This read-only property lists the parent of this document element.

`Style` — Format specification

array of format objects

Format objects that specify the format of a document element.

`StyleName` — Style to apply from stylesheet

string

Name of the style to apply from the stylesheet, specified as a string.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

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

More About

- “Report Formatting Approaches” on page 13-21

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 a `mlreportgen.dom.Color` object.

Input Arguments

`type` — Style of underline

string

String specifying the style of the underline. HTML and PDF support only `'single'`.

String	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

String	Description
`'dashed'`	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
`'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

string

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

See Also

mlreportgen.dom.Text

More About

- “Report Formatting Approaches” on page 13-21

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 from a cell array of strings specifying the list items.

Input Arguments

`items` — Content to include in each item in unordered list

cell array of strings

A one-dimensional cell array containing a string for each item in the unordered list.

The cell array can contain a combination of the following:

- A string
- 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 `listItems` argument.

Output Arguments

`unorderedListObj` — Content to include in each item in the 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

The output format must support the custom attributes of this document element.

`Id` — ID for document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Style` — Format specification
array of format objects

Format objects that specify the format of a document element.

`Stylename` — List style name
string

The style specified by `styleName` must be defined in the template used to create the document element to which this list is appended.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

Methods

Method	Purpose
append Use the `UnorderedList.append` method similar to how you use `OrderedList.append`.	Append items to this list.
clone Use the `UnorderedList.clone` method similar to how you use `Paragraph.clone`.	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');
```

- “Create and Format Lists” on page 13-63

See Also

mlreportgen.dom.ListItem | mlreportgen.dom.ListStyleType | mlreportgen.dom.OrderedList

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'`

String that specifies the vertical alignment of a document element.

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Specify vertical alignment

`'top'` (default) | `'bottom'` | `'middle'`

String that specifies the vertical alignment of an object. Choices are:

See Also

mlreportgen.dom.HAlign

More About

- “Report Formatting Approaches” on page 13-21

mlreportgen.dom.VerticalAlign class

Package: mlreportgen.dom

Vertical alignment of text

Description

Specifies vertical alignment of text.

Construction

`verticalAlignObj = VerticalAlign()` creates a superscript alignment.

`verticalAlignObj = VerticalAlign(align)` creates an alignment of the specified type.

Input Arguments

`align` — Vertical alignment of text relative to baseline

`'superscript'` (default) | `'subscript'` | `'baseline'`

String specifying the vertical alignment of text relative to the baseline.

Output Arguments

`verticalAlignObj` — Vertical alignment of text

`mlreportgen.dom.VerticalAlignment` object

Vertical alignment of text, represented by an `mlreportgen.dom.VerticalAlignment` object.

Properties

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Vertical alignment of text relative to baseline

`'superscript'` (default) | `'subscript'` | `'baseline'`

String specifying the vertical alignment of text relative to the baseline.

Examples

Use a Superscript

```
import mlreportgen.dom.*;
doctype = 'html';
d = Document('test',doctype);

p = Paragraph('e = mc');
t = Text('2');
t.Style = {VerticalAlign('superscript')};
append(p,t);
append(d,p);

close(d);
rptview('test',doctype);
```

See Also

mlreportgen.dom.Text

More About

- "Report Formatting Approaches" on page 13-21

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
string

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Source` — Source DOM object from which message originates

a DOM object

Source DOM object from which the message originates.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Text` — Text of message

string

Message text, specified as a string.

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.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);
```

- “Display Report Generation Messages” on page 13-119

See Also

`mlreportgen.dom.MessageDispatcher.dispatch`

mlreportgen.dom.WhiteSpace class

Package: mlreportgen.dom

White space type

Description

Specifies behavior for white space and line breaks in text.

Construction

`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 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.

String	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.

String	Description
`'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.

Output Arguments

`ws` — White space type

`mlreportgen.dom.WhiteSpace` object

White space type, returned as an `mlreportgen.dom.WhiteSpace` object.

Properties

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`WhiteSpace` — White space and line breaks in text

[ ] (default) | string

To specify how to handle white space, use one of these values.

String	Description	Supported Output Types
`normal` (default)	For HTML and PDF, removes spaces at beginning and end of text string. Multiple spaces within string collapse to single space. For Word, removes spaces at beginning and end of text string.	All output types
`nowrap`	Sequences of white space collapse into a single white space. Text never wraps to the next line.	HTML
`preserve`	Preserves spaces and line feeds. Acts like the `<pre>` tag in HTML.	All output types
`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.	HTML and PDF
`pre-wrap`	Preserves white space. Text wraps when necessary and on line breaks	HTML and PDF

Setting the `WhiteSpace` property adds a corresponding `WhiteSpace` format object to `Style` property. Removing the `WhiteSpace` property setting removes the `WhiteSpace` 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 string `'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.Paragraph | mlreportgen.dom.Text

More About

- “Report Formatting Approaches” on page 13-21

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`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

More About

- “Report Formatting Approaches” on page 13-21

mlreportgen.dom.Width class

Package: mlreportgen.dom

Object width

Description

Specifies the width of an object, such as an image or a table cell.

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

string

Width of object, such as an image or a table cell, specified as a string. The string having the format `valueUnits`, where `Units` is an abbreviation for the units in which the width is expressed. The following abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels
- `%` — percent

Output Arguments

`widthObj` — Object width

`mlreportgen.dom.Width` object

Object width, represented by an `mlreportgen.dom.Width` object.

Properties

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Width of object

string

Width of object, such as an image or a table cell, specified as a string. The string having the format `valueUnits`, where `Units` is an abbreviation for the units in which the width is expressed. The following abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas

- `pt` — points
- `px` — pixels
- `%` — percent

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.Image | mlreportgen.dom.Table

More About

- “Report Formatting Approaches” on page 13-21

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 hexidecimal RGB color value.

Input Arguments

`color` — Background color

string

Background color, specified as a string. You can use:

- The name of a color, specified as a string. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- Hexidecimal RGB (red, green, blue) color value, specified as a string. Use a string having the format `#RRGGBB`. Use `#` as the first character and two-digit hexidecimal numbers for each the red, green, and blue value. 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 PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

`Value` — CSS color name or hexadecimal RGB value for this color

string

The name of a color, specified as a string, using one of these values:

- A CSS color name. See http://www.crockford.com/wrrrld/color.html.

- An RGB value, using a string having the format `#RRGGBB`. Use # as the first character and two-digit hexidecimal numbers for each the red, green, and blue value. 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.*
slidesFile = 'myBackground.pptx';
slides = Presentation(slidesFile);
slide1 = add(slides,'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);
```

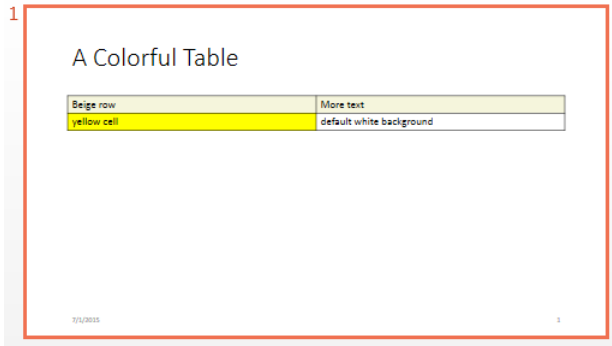
```
append(row2,row2entry2);

append(table1,row1);
append(table1,row2);
```

Replace the slide content with the table, generate the presentation, and open the `myBackground` presentation. (The `winopen` code works on Windows platforms.)

```
replace(slide1,'Content',table1);
close(slides);

if ispc
    winopen(slidesFile);
end
```



- “Create and Format Tables” on page 14-87

See Also

mlreportgen.ppt.FontColor

More About

- “Presentation Formatting Approaches” on page 14-20

mlreportgen.ppt.Bold class

Package: mlreportgen.ppt

Bold for text object

Description

Specifies whether to use bold for a text object.

Construction

`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 string. A setting of `false` (or `0`) uses regular weight text. A setting of `true` (or `1`) renders text in bold.

Data Types: `logical`

Output Arguments

`boldObj` — Bold text

`mlreportgen.ppt.Bold` object

Bold text, returned as an `mlreportgen.ppt.Bold` object.

Properties

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier 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.*

slidesFile = 'myBoldPresentation.pptx';
slides = Presentation(slidesFile);

titleSlide = add(slides,'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);
```

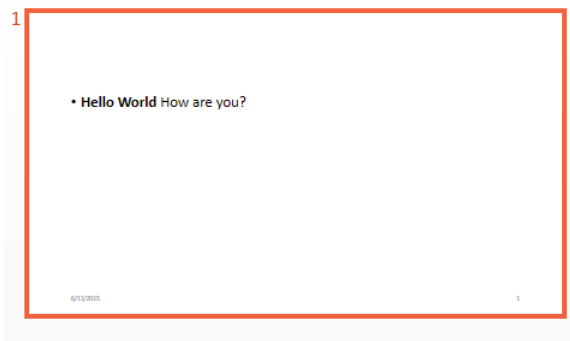
Add the paragraph to the slide and close the presentation.

```
replace(titleSlide,'Content',p);

close(slides);
```

Open `myBoldPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```



- “Create and Format Text” on page 14-81

See Also

mlreportgen.ppt.FontColor | mlreportgen.ppt.FontFamily | mlreportgen.ppt.FontSize | mlreportgen.ppt.Italic

More About

- “Presentation Formatting Approaches” on page 14-20

Introduced in R2015b

mlreportgen.ppt.ColSpec class

Package: mlreportgen.ppt

Formatting for table column

Description

Formatting for a table column.

Construction

`colSpecObj = ColSpec()` creates an empty table column specification.

`colSpecObj = ColSpec(colWidth)` creates a column specification having the specified width.

Input Arguments

`colWidth` — Width of column

string

Width of column, specified as a string. Use a string having the format `valueUnits`, where `Units` is an abbreviation for the width units. These abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

Output Arguments

`colSpecObj` — Table column formatting

`mlreportgen.ppt.ColSpec` object

Table column formatting, returned as an `mlreportgen.ppt.ColSpec` object.

Properties

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

`Width` — Column width

string

Width of table column, specified as a string.

The string 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`Bold` — Option to use bold for text

logical value

Option to use bold for text, specified as a logical. To make text bold, set this property to `true` or `1`. Setting the `Bold` property adds a corresponding `mlreportgen.ppt.Bold` format object to the `Style` property of this presentation element. Removing the `Bold` property setting removes the object.

Data Types: `logical`

`Font` — Default font for text in column

string

Default font for text in column, specified as a string. Specify a font that appears in the PowerPoint list of fonts in the **Home** tab **Font** area.

`ComplexScriptFont` — Font family for complex scripts

string

Font family for complex scripts, specified as a string. Specify a font family for substituting in a locale that requires a complex script (such as Arabic or Asian) for rendering text.

`FontColor` — Font color for presentation element

string

Font color, specified as a string. Use either a CSS color name or a hexadecimal RGB value.

- For a list of CSS color names, see http://www.crockford.com/wrrrld/color.html.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexidecimal numbers for each for the red, green, and blue values. For example, `'#0000ff'` specifies blue.

`FontSize` — Font size

string

Font size, specified as a string. Use the string format `valueUnits`, where `Units` is an abbreviation for the font size. These abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters

- `pi` — picas
- `pt` — points
- `px` — pixels

`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`

Examples

Set Table Column Formatting

Create a presentation and add a slide.

```
import mlreportgen.ppt.*
slides = Presentation('myColSpec.pptx');
tableSlide = add(slides,'Title and Content');
```

Create a table. Create a `ColSpec` object with a specified width for the first two columns of the table. Specify the `BackgroundColor` property for the two `ColSpec` objects. Set the `ColSpecs` property of the `Table` object `t` to the `colSpecs`, which specifies the formatting for the first two columns.

```
t = Table(magic(12));
t.Style = {HAlign('center')};

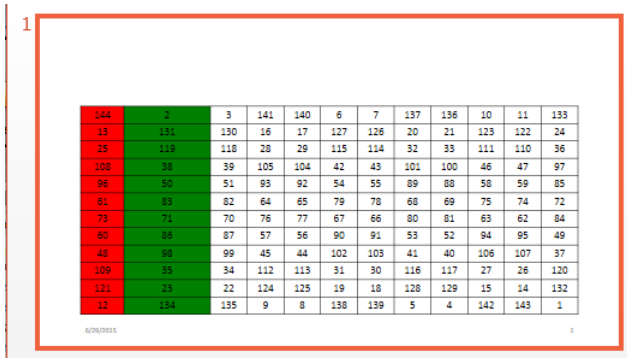
colSpecs(2) = ColSpec('2in');
colSpecs(1) = ColSpec('1in');
colSpecs(1).BackgroundColor = 'red';
colSpecs(2).BackgroundColor = 'green';
t.ColSpecs = colSpecs;
```

Add the table to the slide, generate the presentation, and open the `myColSpec` presentation. (The `winopen` code works on Windows platforms.)

```
replace(slides,'Content',t);
```

```
close(slides);

if ispc
    winopen(slidesFile);
end
```



- “Create and Format Tables” on page 14-87

See Also

mlreportgen.ppt.ColWidth

More About

- “Presentation Formatting Approaches” on page 14-20

mlreportgen.ppt.ColWidth class

Package: mlreportgen.ppt

Table column width

Description

Width of a table column.

Construction

`widthObj = ColWidth()` creates a format object that specifies a column width of one inch.

`widthObj = ColWidth(value)` creates a column width object having the specified width.

Input Arguments

`width` — Width of column

string

Width of column, specified as a string. Use a string having the format `valueUnits`, where `Units` is an abbreviation for the width units. These abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

Output Arguments

`widthObj` — Column width

`mlreportgen.ppt.ColWidth` object

Column width, returned as an `mlreportgen.ppt.ColWidth` object.

Properties

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

`Width` — Width of column

string

Width of column, specified as a string. Use a string having the format `valueUnits`, where `Units` is an abbreviation for the width units. These abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

Examples

Set Table Column Width

Create a presentation.

```
import mlreportgen.ppt.*
slides = Presentation('myColWidth.pptx');
slide1 = add(slides,'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. (The `winopen` code works on Windows platforms.)

```
replace(slide1,'Content',t);
close(slides);

if ispc
    winopen(slidesFile);
end
```

- “Create and Format Tables” on page 14-87

See Also

mlreportgen.ppt.ColSpec | mlreportgen.ppt.Table

More About

- “Presentation Formatting Approaches” on page 14-20

mlreportgen.ppt.ContentPlaceholder class

Package: mlreportgen.ppt

Placeholder for presentation content in slide

Description

Placeholder for replaceable content in a slide. Represents slide content that can be replaced by a paragraph, picture, or table. It can also represent slide content that includes one or more paragraphs. Use the PowerPoint editor to create content placeholders in a PowerPoint template or presentation for content that you want to replace programmatically.

You can use a `Presentation.find` or `Slide.find` method to find content placeholders in a presentation or a slide. The `find` method returns the content placeholders as instances of this class. You can also access slide content placeholders by accessing its children using its `Children` property. If a child is an object of this type, you can replace it with a paragraph, picture, or table, using the `replac`e method of the type of object that you want to replace. If the content placeholder object is empty or already contains paragraphs, you can add paragraphs to the content, using the `ContentPlaceholder.add` method.

The text format properties of the placeholder object specify the default format for content that replaces the placeholder.

Properties

`Bold` — Option to use bold for paragraphs contained in placeholder

[ ] (default) | logical value

Option to use bold for paragraphs contained in placeholder, specified as a logical value. This property sets the default format, which you can override for specific `Paragraph` objects. Setting the `Bold` property adds a corresponding `mlreportgen.ppt.Bold` format object to the `Style` property of this presentation element. Removing the `Bold` property setting removes the object.

Data Types: `logical`

`FontColor` — Font color for paragraphs included in placeholder

string

Font color for paragraphs included in placeholder, specified as a string. Use either a CSS color name or a hexadecimal RGB value. This property sets the default format, which you can override for specific `Paragraph` objects.

- For a list of CSS color names, see http://www.crockford.com/wrrrld/color.html.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexidecimal numbers for each the red, green, and blue values. For example, `'#0000ff'` specifies blue.

`Italic` — Option to use italics for paragraphs included in placeholder

[ ] (default) | logical value

Option to use italics for paragraphs included in placeholder, specified as a logical value. This sets the default format, which you can override for specific Paragraph objects. 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` — Option to underline paragraphs included in placeholder

[ ] (default) | logical value

Option to underline paragraphs included in placeholder, specified as a string. This property sets the default format, which you can override for specific `Paragraph` objects. You can specify one of these types of underlines.

String	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

String	Description
'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

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.

`Name` — Content placeholder name

string

Content placeholder name, specified as a string.

`X` — Upper-left x coordinate position of placeholder content

string

Upper-left x coordinate of placeholder content, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the units in which the x position is expressed. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Y` — Upper-left y coordinate position of placeholder content

string

Upper-left y coordinate of placeholder content, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the units in which the y position is expressed. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Width` — Width of placeholder

string

Width of placeholder, specified as a string.

The string 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`Height` — Height of placeholder

string

Height of placeholder, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the units in which the height is expressed. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Style` — Formatting for paragraphs included in placeholder

cell array of PPT style objects

Formatting for the paragraphs included in placeholder, specified as an array of PPT style objects. This sets the default format, which you can override for specific Paragraph objects. You can specify these `mlreportgen.ppt` style 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 `mlreportgen.ppt.Element` objects

This read-only property lists child elements that the object contains, specified as a cell array.

`Parent` — Parent of this PPT API object

PPT object

This read-only property lists the parent of this object, specified a PPT object.

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

Methods

Method	Purpose
`add`	Add content to the placeholder.
`replace`	Replace placeholder with the content.

Examples

Replace Content Using a Content Placeholder

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myPlaceholderPresentation.pptx';
slides = Presentation(slidesFile);

add(slides,'Title and Content');
add(slides,'Title and Content');

close(slides);

% Open myPlaceholderPresentatiopresentation.pptx.
% On Windows platforms, you can run this code.

if ispc
    winopen(slidesFile);
end
```

In the `myPlaceholderPresentation.pptx` presentation file, In the **Home** tab, select **Select** > **Selection Pane**.

Select the first slide, place the cursor in the `Click to add text` and type `Topic 1`.

In the **Selection** pane, the placeholder is called `Content`.

Change the `Content` object name to `Agenda`.

In the PowerPoint editor, save and close the presentation.

In MATLAB, create a presentation that uses `myPlaceholderPresentation` as the presentation template.

```
slides = Presentation('myPlaceholderPresentation','myPlaceholderPresentation');
```

Use the `mlreportgen.ppt.Presentation.find` method to find the slides that have an `Agenda` placeholder. In this case, there is only one.

```
contents = find(slides,'Agenda')
```

```
ContentPlaceholder with properties:

contents =

          Bold: []
     FontColor: []
        Italic: []
        Strike: []
     Subscript: []
   Superscript: []
     Underline: []
          Name: 'Agenda'
             X: []
             Y: []
         Width: []
        Height: []
         Style: []
```

```
        Children: []
          Parent: [1x1 mlreportgen.ppt.Slide]
             Tag: 'ppt.ContentPlaceholder:11127:4401'
              Id: '11127:4401'
```

Add text to the replace the content placeholder in the first slide. Make the text bold.

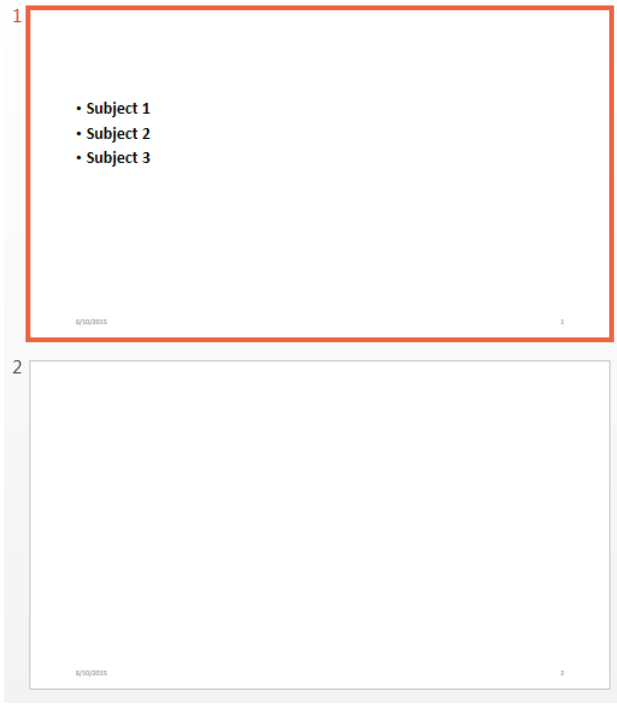
```
replace(contents(1),{'Subject 1','Subject 2','Subject 3'});
contents(1).Bold = true;
```

Close the presentation to generate the output.

```
close(slides);

if ispc
    winopen(slidesFile);
end
```

Open `myPlaceholderPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:



The bullets in the first slide reflect the replaced content for the Agenda placeholder.

- “Access PowerPoint Template Elements” on page 14-37
- “Add and Replace Presentation Content” on page 14-74

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
string

The text to display for the message, specified as a string.

`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 PPT API object
string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

`Text` — Text of the message

string

Message text, specified as a string.

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);
```

- “Display Presentation Generation Messages” on page 14-15

See Also

mlreportgen.ppt.MessageDispatcher.dispatch | mlreportgen.ppt.MessageEventData

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
string

The text to display for the message, specified as a string.

`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 PPT API object
string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier 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

string

Message text, specified as a string.

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);
```

- “Display Presentation Generation Messages” on page 14-15

See Also

mlreportgen.ppt.MessageDispatcher.dispatch | mlreportgen.ppt.MessageEventData

Introduced in R2015b

mlreportgen.ppt.ExternalLink class

Package: mlreportgen.ppt

Hyperlink to location outside of presentation

Description

Defines a hyperlink to a location outside of the presentation.

Construction

`externalLinkObj = ExternalLink()` creates an empty external link object.

`externalLinkObj = ExternalLink(target,linkText)` creates a hyperlink with the specified link text.

Input Arguments

`target` — URL of link target
string

URL of the link target, specified as a string.

`linkText` — Link text
string

The text to use for the link text.

Output Arguments

`externalLinkObj` — External link
`mlreportgen.ppt.ExternalLink` object

External link, returned as an `mlreportgen.ppt.ExternalLink` object.

Properties

`Target` — URL of link target
string

URL of the link target, specified as a string. Specify the full URL (for example, include `http://`).

`Content` — Link text
string

The text to use for the link text.

`Style` — Text formatting for external link
cell array of PPT style objects

Format for the external link text, specified as an array of PPT style objects. You can specify these `mlreportgen.ppt` style objects:

- `FontFamily` object
- `FontSize` object
- `Bold` object
- `FontColor` object
- `Italic` object
- `Underline` object

`Id` — ID for PPT API object
string

ID for PPT API object, specified as a string. 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
session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

`Target` — Target URL of link

string

This read-only property displays the URL of the link target of this hyperlink.

Examples

Add External Link to Presentation

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myExternalLinkPresentation.pptx';
slides = Presentation(slidesFile);

add(slides,'Title and Content');
```

Create a `Paragraph` object and an `ExternalLink` object.

```
p = Paragraph('This is a link to the');
link = ExternalLink('http://www.mathworks.com',' MathWorks site.');
```

Append the external link to the paragraph, and replace the `Content` placeholder.

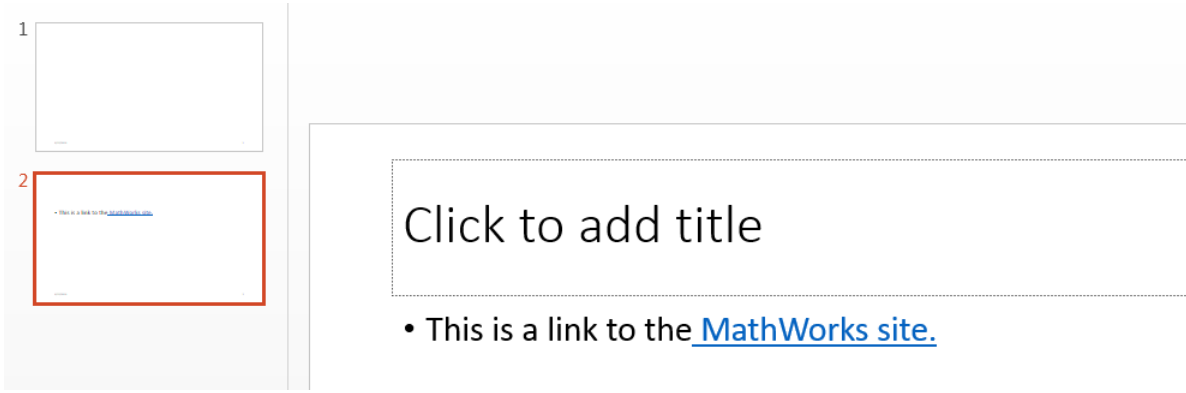
```
append(p,link);
replace(slides,'Content',p);
```

Close the presentation.

```
close(slides);
```

Open `myExternalLinkPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```



- “Create and Format Links” on page 14-98

See Also

mlreportgen.ppt.Paragraph

mlreportgen.ppt.FontColor class

Package: mlreportgen.ppt

Font color of presentation element

Description

Specifies the font color of a presentation element.

Construction

`colorObj = FontColor()` creates a black font color object.

`colorObj = FontColor(color)` creates a font color object based on the specified CSS color name.

Input Arguments

`color` — Font color

string

Font color, specified as a string. You can use:

- The name of a color. The name must be a CSS color name. See http://www.crockford.com/wrrrld/color.html.
- Hexadecimal RGB (red, green, blue) color value. Use a string having the format `#RRGGBB`. Use # as the first character and two-digit hexidecimal numbers for each the red, green, and blue value. For example, `'#0000ff'` specifies blue.

Output Arguments

`colorObj` — Font color for presentation element

`mlreportgen.ppt.FontColor` object

Font color for presentation element, returned as an `mlreportgen.ppt.FontColor` object.

Properties

`HexValue` — hexadecimal color value

string

This read-only property specifies a hexadecimal 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 PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

`Value` — CSS color name or hexadecimal RGB value for this color

string

A CSS color name or a hexadecimal RGB value, specified as a string

- For a list of CSS color names, see http://www.crockford.com/wrrrld/color.html.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexadecimal numbers for each the red, green, and blue value. For example, `'#0000ff'` specifies blue.

Examples

Create Colored Text

Create a presentation.

```
import mlreportgen.ppt.*
```

```
slidesFile = 'myFontColorPresentation.pptx';
slides = Presentation(slidesFile);

titleSlide = add(slides,'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 and close the presentation.

```
replace(titleSlide,'Content',p);

close(slides);
```

Open `myFontColorPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```

- “Create and Format Text” on page 14-81

See Also

mlreportgen.ppt.Bold | mlreportgen.ppt.FontFamily | mlreportgen.ppt.FontSize | mlreportgen.ppt.Italic

More About

- “Presentation Formatting Approaches” on page 14-20

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

string

Font family, specified as a string. Specify a font that appears in the PowerPoint list of fonts in the **Home** tab **Font** area.

`complexScriptFont` — Font family for complex scripts

string

Font family for complex scripts, specified as a string. 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

string

Font family for complex scripts, specified as a string. Specify a font family for substituting in a locale that requires a complex script (such as Arabic) for rendering text.

`Font` — Font family

string

Font family, specified as a string. Specify a font that appears in the PowerPoint list of fonts in **Home** tab **Font** area.

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

Examples

Set the Font Family

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myFontFamilyPresentation.pptx';
slides = Presentation(slidesFile);
```

```
titleSlide = add(slides,'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);
```

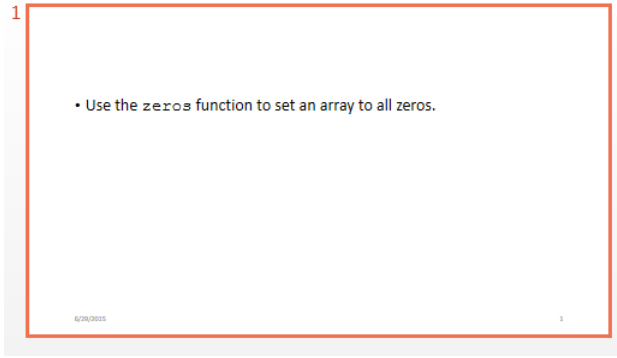
Add the paragraph to the slide and close the presentation.

```
replace(titleSlide,'Content',p);

close(slides);
```

Open `myFontFamilyPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```



- “Create and Format Text” on page 14-81

- “Create and Format Paragraphs” on page 14-84

See Also

mlreportgen.ppt.FontColor | mlreportgen.ppt.FontSize | mlreportgen.ppt.Paragraph | mlreportgen.ppt.Text

More About

- “Presentation Formatting Approaches” on page 14-20

Introduced in R2015b

mlreportgen.ppt.FontSize class

Package: mlreportgen.ppt

Font size

Description

Specifies the size of a font.

Construction

`fontSizeObj = FontSize()` an empty font size object.

`fontSizeObj = FontSize(sizeStr)` specifies the specified font size.

Input Arguments

`sizeStr` — Font size
string

Font size, specified as a string. Use the string format `valueUnits`, where `Units` is an abbreviation for the units in which the size is expressed. These abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

Output Arguments

`fontSizeObj` — Font size
`mlreportgen.ppt.FontSize` object

Font size, returned as an `mlreportgen.ppt.FontSize` object.

Properties

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

`Value` — Font size

`'12pt'` (default) | string

Font size, specified as a string. Use the string format `valueUnits`, where `Units` is an abbreviation for the units in which the size is expressed. These abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

Examples

Set the Font Size

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myFontSizePresentation.pptx';
slides = Presentation(slidesFile);

titleSlide = add(slides,'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);

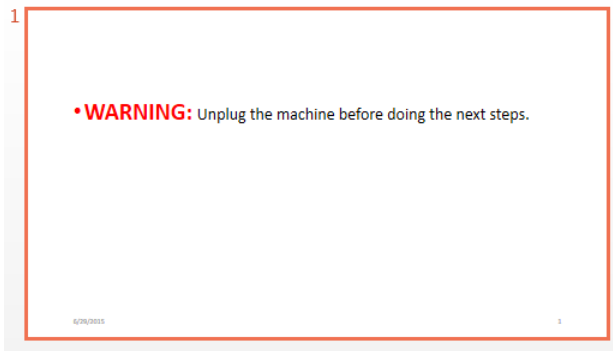
tDesc = Text(' Unplug the machine before doing the next steps.');
append(p,tDesc);
```

Add the paragraph to the slide, generate the presentation, and open `myFontFamilyPresentation`. (The `winopen` code works on Windows platforms.)

```
replace(titleSlide,'Content',p);

close(slides);

if ispc
    winopen(slidesFile);
end
```



- “Create and Format Text” on page 14-81
- “Create and Format Paragraphs” on page 14-84

See Also

mlreportgen.ppt.FontColor | mlreportgen.ppt.FontFamily | mlreportgen.ppt.Paragraph | mlreportgen.ppt.Text

More About

- “Presentation Formatting Approaches” on page 14-20

Introduced in R2015b

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.

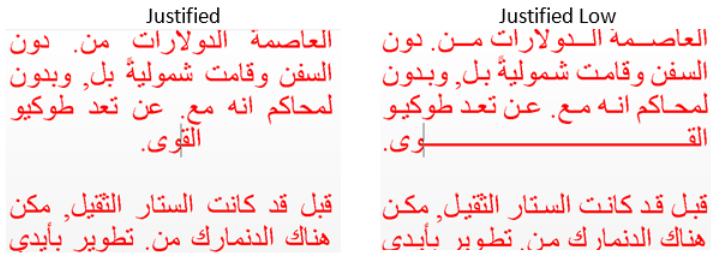
Input Arguments

`value` — Horizontal alignment

string

Horizontal alignment, specified as one of these values:

- `'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



Output Arguments

`horizontalAlignObj` — Horizontal alignment

`mlreportgen.ppt.HAlign` object

Horizontal alignment, returned as an `mlreportgen.ppt.HAlign` object.

Properties

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

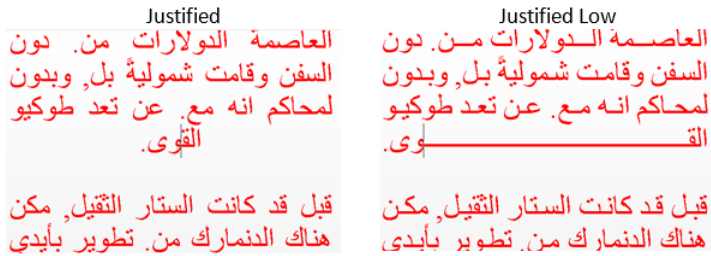
An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

`Value` — Horizontal alignment

string

Horizontal alignment, specified as one of these values:

- `'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 text, adding extra spaces between characters for languages with tone and vowel marks
- `'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.*
slidesFile = 'myHAlignPresentation.pptx';
slides = Presentation(slidesFile);
titleSlide = add(slides,'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`. (The `winopen` code works on Windows platforms.)

```
replace(titleSlide,'Title',p);

close(slides);

if ispc
    winopen(slidesFile);
end
```

MATLAB:

- “Create and Format Paragraphs” on page 14-84

See Also

mlreportgen.ppt.Paragraph | mlreportgen.ppt.VAlign

More About

- “Presentation Formatting Approaches” on page 14-20

Introduced in R2015b

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 PPT API object
string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier 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.*

slidesFile = 'myItalicPresentation.pptx';
slides = Presentation(slidesFile);

titleSlide = add(slides,'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);
```

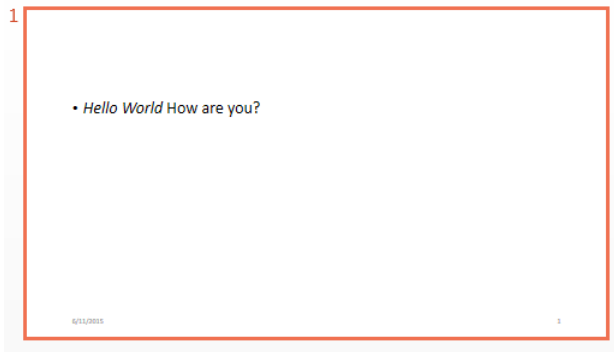
Add the paragraph to the slide and close the presentation.

```
replace(titleSlide,'Content',p);

close(slides);
```

Open `myItalicPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```



- “Create and Format Text” on page 14-81

See Also

mlreportgen.ppt.Bold

More About

- “Presentation Formatting Approaches” on page 14-20

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

string

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 PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

Methods

Method	Purpose
`dispatch`	Dispatch a presentation generation status message.
`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);
```

- “Display Presentation Generation Messages” on page 14-15

See Also

mlreportgen.ppt.MessageDispatcher.dispatch | mlreportgen.ppt.MessageDispatcher.getTheDispatcher | mlreportgen.ppt.MessageEventData | mlreportgen.ppt.MessageFilter

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 PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier 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);
```

- “Display Presentation Generation Messages” on page 14-15

See Also

mlreportgen.ppt.MessageDispatcher.dispatch

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.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. Then open the presentation.

```
titleText = Text('This is a Title');
titleText.Style = {Bold};
replace(pre,'Title',titleText);

close(pre);
```

- “Display Presentation Generation Messages” on page 14-15

See Also

mlreportgen.ppt.MessageDispatcher.dispatch | mlreportgen.ppt.MessageEventData

Introduced in R2015b

mlreportgen.ppt.Paragraph class

Package: mlreportgen.ppt

Formatted block of text (paragraph)

Description

To define a paragraph, use an `mlreportgen.ppt.Paragraph` object. A paragraph can contain text strings, an `mlreportgen.ppt.Text` objects, or `mlreportgen.ppt.ExternalLink` objects.

Construction

`paragraphObj = Paragraph()` creates an empty paragraph object.

`paragraphObj = Paragraph(text)` creates a paragraph containing a `mlreportgen.ppt.Text` object with the text specified by the `text` string.

`paragraphObj = Paragraph(pptElementObj)` creates a paragraph containing the text or external link specified by `pptElementObj`.

Input Arguments

`text` — Paragraph text
string

Paragraph text, specified as a string.

`pptElementObj` — Presentation element to include in paragraph
`mlreportgen.ppt.Text` object | `mlreportgen.ppt.ExternalLink` object

Presentation element to include in paragraph, specified as either an `mlreportgen.ppt.Text` or `mlreportgen.ppt.ExternalLink` object.

Output Arguments

`paragraphObj` — Paragraph
`mlreportgen.ppt.Paragraph` object

Paragraph, returned as an `mlreportgen.ppt.Paragraph` object.

Properties

`Bold` — Option to use bold for text

logical value

Option to use bold for text, specified as a logical. To make text bold, set this property to `true` or `1`. Setting the `Bold` property adds a corresponding `mlreportgen.ppt.Bold` format object to the `Style` property of this presentation element. Removing the `Bold` property setting removes the object.

Data Types: `logical`

`FontColor` — Font color for presentation element

string

Font color, specified as a string. Use either a CSS color name or a hexadecimal RGB value.

- For a list of CSS color names, see http://www.crockford.com/wrrrld/color.html.
- To specify a hexadecimal RGB format, use `#` as the first character and two-digit hexidecimal 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) | string

Type of underlining for text, specified as a string. 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.

String	Description
`'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

`Level` — Indentation level of paragraph

string

Indentation level of a paragraph, specified as a string.

- `1` — Top-level paragraph (no indentation)
- `2` — Second-level paragraph
- `3` — Third-level paragraph
- `4` — Fourth-level paragraph

`Style` — Text formatting

cell array of PPT style objects

Text formatting, specified as a cell array of PPT style objects. You can specify these `mlreportgen.ppt` style objects:

- `FontFamily` object
- `FontSize` object
- `Bold` object
- `FontColor` object
- `Italic` object
- `Underline` object

`Children` — Children of this PPT API object

cell array of `mlreportgen.ppt.Element` objects

This read-only property lists child elements that the object contains, specified as a cell array.

`Parent` — Parent of this PPT API object

PPT object

This read-only property lists the parent of this object, specified a PPT object.

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

Methods

Method	Purpose
`append`	Append text or external link to paragraph.

Examples

Add Paragraphs to Presentation

Create a presentation with two slides.

```
import mlreportgen.ppt.*;

slidesFile = 'myParagraphPresentation.pptx';
slides = Presentation(slidesFile);

add(slides,'Title Slide');
add(slides,'Title and Content');
```

Create a `Paragraph` object to use for the title of slides. 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(slides,'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.

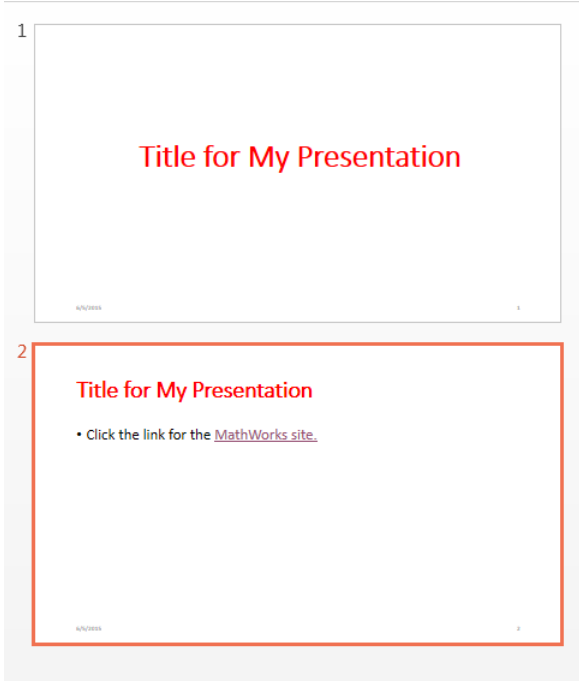
```
replace(slides,'Content',p1);
```

Close the presentation.

```
close(slides);
```

Open `myParagraphPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```



- “Create and Format Paragraphs” on page 14-84

See Also

mlreportgen.ppt.ContentPlaceholder | mlreportgen.ppt.Text | mlreportgen.ppt.TextBox

mlreportgen.ppt.Picture class

Package: mlreportgen.ppt

Create picture to include in presentation

Description

Create a picture to include in a presentation.

Construction

`pictureObj = Picture()` creates an empty picture object.

`pictureObj = Picture(picturePath)` creates a picture object containing the picture specified by `picturePath`.

Input Arguments

`picturePath` — Path of picture file

string

Path of a picture file, specified as a string. Use one of these formats (you cannot use `.svg` format):

- `.bmp`
- `.emf`
- `.eps`
- `.gif`
- `.jpeg`
- `.jpg`
- `.png`
- `.tif`
- `.tiff`

Output Arguments

`pictureObj` — Picture
`mlreportgen.ppt.Picture` object

Picture, returned as an `mlreportgen.ppt.Picture` object.

Properties

`Name` — Picture name
string

Picture name, specified as a string.

`X` — Upper-left x-coordinate position of picture
string

Upper-left x-coordinate position of picture, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the x-position units. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Y` — Upper-left y-coordinate position of picture
string

Upper-left y-coordinate position of picture, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the y-position units. Valid abbreviations are:

- No abbreviation — pixels

- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Width` — Width of picture

string

Width of picture, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the units for the width. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Height` — Height of picture

string

Height of picture, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the units for the height. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas

- `pt` — points
- `px` — pixels

`Style` — Picture placeholder formatting

ignored

Picture placeholder formatting. This property is ignored.

`Children` — Children of this PPT API object

cell array of `mlreportgen.ppt.Element` objects

This read-only property lists child elements that the object contains, specified as a cell array.

`Parent` — Parent of this PPT API object

PPT object

This read-only property lists the parent of this object, specified a PPT object.

`Tag` — Tag for this PPT API object

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

Methods

Method	Purpose
`replace`	Replace picture with another picture.

Examples

Add Picture to Presentation

Create a presentation with two slides.

```
import mlreportgen.ppt.*

slidesFile = 'myPicturePresentation.pptx';
slides = Presentation(slidesFile);

add(slides,'Title Slide');
add(slides,'Title and Content');
```

Create a `Picture` object using an airplane image available in MATLAB. Specify the size for the picture.

```
plane = Picture(which('b747.jpg'));
plane.Width = '5in';
plane.Height = '2in';
```

Replace the content of the second slide with the `plane` picture.

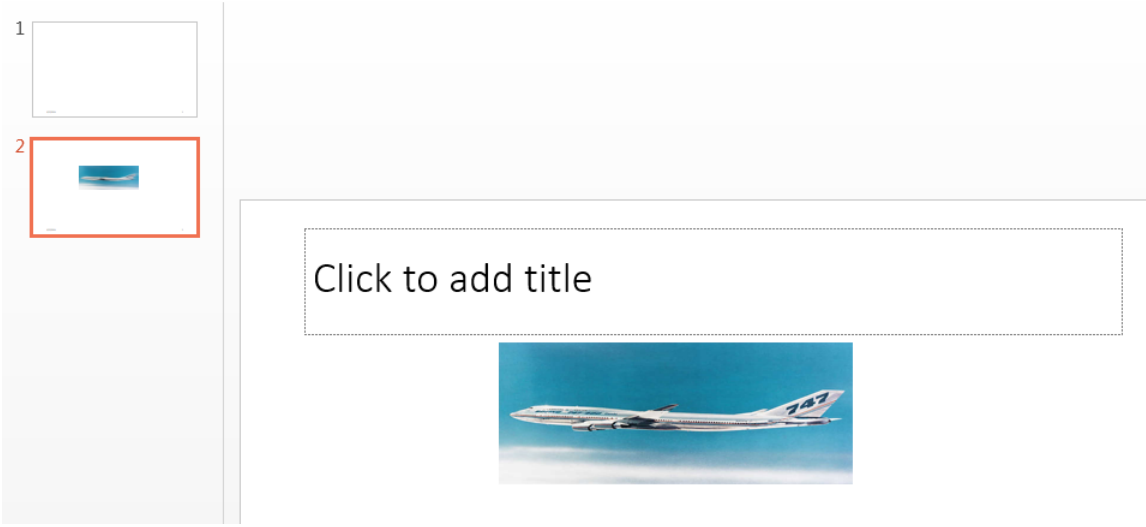
```
replace(slides,'Content',plane);
```

Close the presentation.

```
close(slides);
```

Open `myPicturePresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```



- “Create and Format Pictures” on page 14-96

See Also

mlreportgen.ppt.PicturePlaceholder

mlreportgen.ppt.PicturePlaceholder class

Package: mlreportgen.ppt

Slide placeholder to replace with picture

Description

Slide placeholder to replace with a picture. You can create a picture placeholder using a layout slide. In the default PPT API, when you add a `Title and Picture` slide to a presentation, the API creates a `PicturePlaceholder` object. Use the `find` method with the `Slide` object to find the picture placeholder. You can then set properties for that `PicturePlaceholder` object.

Use the PowerPoint editor to insert a picture placeholder in a presentation.

1. Select the **Slide Master** tab.
2. Click the layout slide you want to add the picture placeholder to. You can add the placeholder to an existing or a new layout slide.
3. In the toolbar, click **Insert Placeholder** and select **Picture**.

Properties

`Name` — Picture placeholder name

string

Picture placeholder name, specified as a string.

`X` — Upper-left x-coordinate position of picture placeholder

string

Upper-left x-coordinate of picture placeholder, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the x-position units. Valid abbreviations are:

- No abbreviation — pixels

- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Y` — Upper-left y-coordinate position of picture placeholder

string

Upper-left y-coordinate position of picture placeholder, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the y-position units. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Width` — Width of placeholder

string

Width of placeholder, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the units for the width. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas

- `pt` — points
- `px` — pixels

`Height` — Height of placeholder
string

Height of placeholder, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the units for the height. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Style` — Picture placeholder formatting
ignored

Picture placeholder formatting. This property is ignored.

`Children` — Children of this PPT API object
cell array of `mlreportgen.ppt.Element` objects

This read-only property lists child elements that the object contains, specified as a cell array.

`Parent` — Parent of this PPT API object
PPT object

This read-only property lists the parent of this object, specified a PPT object.

`Id` — ID for PPT API object
string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

Methods

Method	Purpose
`replace`	Replace picture placeholder with picture

Examples

Replace a Picture

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myPicturePlaceholderPresentation.pptx';
slides = Presentation(slidesFile);
slide1 = add(slides,'Title and Picture');
```

Create an `mlreportgen.ppt.Picture` object.

```
plane = Picture(which('b747.jpg'));
plane.X = '1in';
plane.Y = '1in';
plane.Width = '3in';
plane.Height = '2in';
```

The default PPT API `Title and Picture` layout slide has a `Picture` object that is a picture placeholder. Use the `mlreportgen.ppt.Slide.find` method to find an object with the `Name` property of `Picture`.

```
contents = find(slide1,'Picture');
```

Replace the picture placeholder with the picture.

```
replace(contents(1),plane);
```

Close the presentation.

```
close(slides);
```

Open `myPicturePlaceholderPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```

- “Access PowerPoint Template Elements” on page 14-37
- “Add or Replace a Picture” on page 14-79

See Also

mlreportgen.ppt.Picture

Introduced in R2015b

mlreportgen.ppt.Presentation class

Package: mlreportgen.ppt

Create Microsoft PowerPoint presentation

Description

Creates a Microsoft PowerPoint presentation.

Construction

`presentationObj = Presentation()` creates a presentation named `Untitled.pptx` in the current folder, using the default PPT API `default.pptx` template.

`presentationObj = Presentation(presentationPath)` creates a presentation at the specified location.

`presentationObj = Presentation(presentationPath,templatePath)` creates a presentation using the PowerPoint template at the specified location.

Input Arguments

`presentationPath` — Path to presentation file
string

Path to the presentation file, specified as a string.

`templatePath` — Path to PowerPoint template for presentation
string

Path to the PowerPoint template for the presentation, specified as a string.

Output Arguments

`presentationObj` — Presentation
`mlreportgen.ppt.Presentation` object

Presentation object, returned as an `mlreportgen.ppt.Presentation` object

Properties

`TemplatePath` — Path of the template used for this presentation

string

The path to the PowerPoint template to use for this presentation element, specified as a string.

`OutputPath` — Path of output file or folder for this presentation

string

You set this property only before you open the presentation.

Path of this output file of the presentation, specified as a string.

`Children` — Children of this PPT API object

cell array of `mlreportgen.ppt.Element` objects

This read-only property lists child elements that the object contains, specified as a cell array.

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

Methods

Method	Purpose
`open`	Open presentation.

Method	Purpose
`close`	Close presentation.
`find`	Search in presentation. Use the `Presentation.find` method the same way you use the `Slide.find` method.
`replace`	Replace content in presentation.
`add`	Add content to presentation.
`getMasterNames`	Get names of slide masters for presentation
`getLayoutNames`	Get names of layouts for presentation.
`getTableStyleNames`	Get table style names for presentation.

Examples

Create Presentation Using Default Template

Create a presentation with three slides. Use the default PPT presentation template.

```
import mlreportgen.ppt.*

slides = Presentation('myFirstPresentation');

add(slides,'Title Slide');
add(slides,'Title and Content');
add(slides,'Title and Picture');
```

Examine the slides `Presentation` object.

```
slides
```

```
slides =

  Presentation with properties:

    TemplatePath: 'matlab/toolbox/shared/mlreportgen/ppt/resources/...'
      OutputPath: 'myFirstPresentation.pptx'
```

```
        Children: [1x3 mlreportgen.ppt.Slide]
          Parent: []
             Tag: 'ppt.Presentation:1181'
              Id: '1181'
```

Examine the first slide.

```
slides.Children(1)

ans =

  Slide with properties:

         Layout: 'Title Slide'
    SlideMaster: 'Office Theme'
           Name: ''
          Style: []
       Children: [1x2 mlreportgen.ppt.TextBoxPlaceholder]
         Parent: [1x1 mlreportgen.ppt.Presentation]
            Tag: 'ppt.Slide:1183'
             Id: '1183'
```

Specify a title for the presentation. Find the `Title` placeholder in the first slide and provide a title. Make the title red.

```
contents = find(slides,'Title');
replace(contents(1),'My First Presentation');
contents(1).FontColor = 'red';
```

Add content to the second slide.

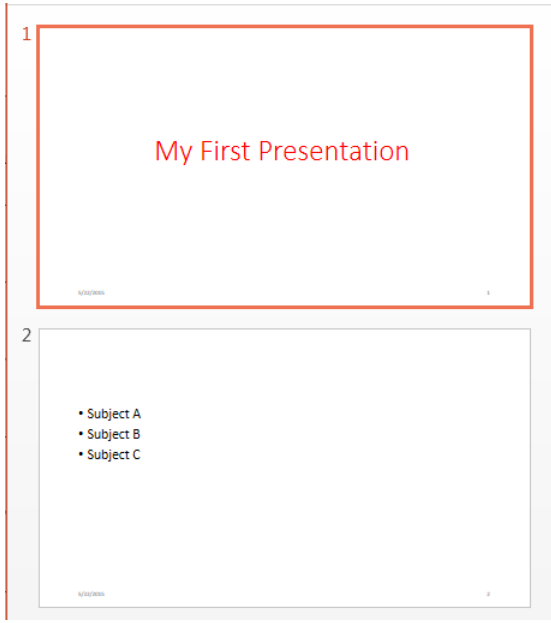
```
contents = find(slides,'Content');
replace(contents,{'Subject A','Subject B','Subject C'});
```

Close the presentation to generate the output.

```
close(slides);
```

Open `myFirstPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```



Create Presentation Specifying a Template

Create a presentation with using the `myFirstPresentation` presentation as the template (see the previous example).

```
import mlreportgen.ppt.*

slides = Presentation('mySecondPresentation','myFirstPresentation');
```

Change the title to `My Second Presentation`.

```
contents = find(slides,'Title');
replace(contents(1),'My Second Presentation');
```

Close the presentation to generate the output.

```
close(slides);
```

Open `mySecondPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
```

```
    winopen(slidesFile);
end
```

- “Create a Presentation Object to Hold Content” on page 14-13
- “Set Up a PowerPoint Template” on page 14-26

See Also

mlreportgen.ppt.Slide

Introduced in R2015b

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

string

The text to display for the message, specified as a string.

`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 PPT API object
string

ID for PPT API object, specified as a string. 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
session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

`Text` — Text of message
string

Message text, specified as a string.

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);
```

- “Display Presentation Generation Messages” on page 14-15

See Also

mlreportgen.ppt.MessageDispatcher.dispatch | mlreportgen.ppt.MessageEventData

Introduced in R2015b

mlreportgen.ppt.Slide class

Package: mlreportgen.ppt

Presentation slide

Description

Presentation slide. To add a slide, use the `mlreportgen.ppt.Presentation.add` method, specifying an output argument. You can then use `mlreportgen.ppt.Slide` methods on that slide object.

Properties

`Layout` — Slide layout

string

This read-only property is the slide layout, specified as a string.

`SlideMaster` — Slide master

string

This read-only property is the slide master, specified by a string.

`Name` — Slide name

string

This read-only property is the slide name, specified as a string.

`Style` — Formatting

cell array of PPT style objects

This read-only property is the slide formatting, represented as a cell array of PPT style objects.

`Children` — Children of this PPT API object

cell array of `mlreportgen.ppt.Element` objects

This read-only property lists child elements that the object contains, specified as a cell array.

`Parent` — Parent of this PPT API object

PPT object

This read-only property lists the parent of this object, specified a PPT object.

`Tag` — Tag for slide

session-unique tag generated as part of object creation (default) | string

This read-only property identifies the slide, specified as a string. 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.

`Id` — ID for slide

string

This read-only property is the session-unique ID generated as part of object creation, specified as a string.

Methods

Method	Purpose
`find`	Search a slide.
`add`	Add content to slide.
`replace`	Replace slide content.

Examples

Create a Slide

Create a presentation.

```
import mlreportgen.ppt.*

slides = Presentation('mySlideAddPresentation.pptx');
slide1 = add(slides,'Title and Picture');
```

Create an `mlreportgen.ppt.Picture` object.

```
plane = Picture(which('b747.jpg'));
plane.X = '4in';
plane.Y = '4in';
plane.Width = '5in';
plane.Height = '2in';
```

Add the plane picture to `slide1`.

```
add(slide1,plane);
```

Close the presentation.

```
close(slides);
```

Open `mySlideAdd.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```

- “Add Slides” on page 14-71
- “Add and Replace Presentation Content” on page 14-74

See Also

mlreportgen.ppt.Presentation

mlreportgen.ppt.Strike class

Package: mlreportgen.ppt

Strike through text

Description

Specifies whether to use a strikethrough line for a text object.

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

string

String specifying the strike type. Choices are:

- `'single'` — Single horizontal line (default)
- `'none'` — No strikethrough line
- `'double'` — Double horizontal line

Output Arguments

`strike` — Strikethrough text

`mlreportgen.ppt.Strike` object

Strikethrough text, returned as an `mlreportgen.ppt.Strike` object.

Properties

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

`Width` — Column width

string

Width of table column, specified as a string.

The string 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`Bold` — Option to use bold for text

logical value

Option to use bold for text, specified as a logical. To make text bold, set this property to `true` or `1`. Setting the `Bold` property adds a corresponding `mlreportgen.ppt.Bold` format object to the `Style` property of this presentation element. Removing the `Bold` property setting removes the object.

Data Types: `logical`

`Font` — Default font for text in column

string

Default font for text in column, specified as a string. Specify a font that appears in the PowerPoint list of fonts in the **Home** tab **Font** area.

`ComplexScriptFont` — Font family for complex scripts

string

Font family for complex scripts, specified as a string. Specify a font family for substituting in a locale that requires a complex script (such as Arabic or Asian) for rendering text.

`FontColor` — Font color for presentation element

string

Font color, specified as a string. Use either a CSS color name or a hexadecimal RGB value.

- For a list of CSS color names, see http://www.crockford.com/wrrrld/color.html.
- To specify a hexadecimal RGB format, use `#` as the first character and two-digit hexidecimal numbers for each for the red, green, and blue values. For example, `'#0000ff'` specifies blue.

`FontSize` — Font size

string

Font size, specified as a string. Use the string format `valueUnits`, where `Units` is an abbreviation for the font size. These abbreviations are valid:

- no abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`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`

`Strike` — Strike type

string

String specifying the strike type. Choices are:

- `'single'` — Single horizontal line (default)
- `'none'` — No strikethrough line
- `'double'` — Double horizontal line

`Subscript` — Display of text as subscript

[ ] (default) | logical value

Display of text as a subscript, specified by a logical value. A setting of `true` (or `1`) renders text as a subscript.

Data Types: `logical`

`Superscript` — Display of text as superscript

[ ] (default) | logical value

Display of text as a superscript, specified by a logical value. A setting of `true` (or `1`) renders text as a superscript.

Data Types: `logical`

`Underline` — Type of underline for text

[ ] (default) | string

Type of underlining for text, specified as a string. 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.

String	Description
`'single'`	Single underline

String	Description
`'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

`BackgroundColor` — Background color

string

The name of a color, specified as a string, using one of these values:

- A CSS color name. See http://www.crockford.com/wrrrld/color.html.
- An RGB value, using a string having the format `#RRGGBB`. Use # as the first character and two-digit hexidecimal numbers for each for the red, green, and blue values. For example, `'#0000ff'` specifies blue.

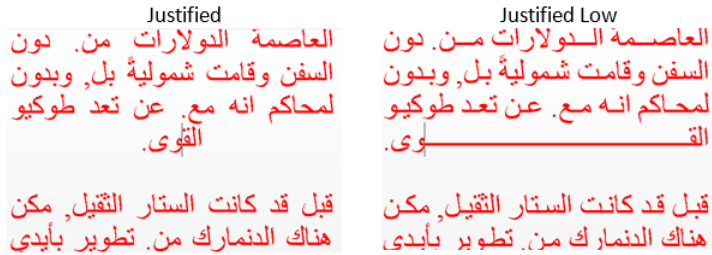
`HAlign` — Horizontal alignment

string

Horizontal alignment, specified as a string.

- `'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



`VAlign` — Vertical alignment for table entry content

`'top'` (default) | string

Vertical alignment for table entry content, specified as a string. Possible values are:

- `'top'`
- `'bottom'`
- `'middle'`
- `'topCentered'`
- `'middleCentered'`
- `'bottomCentered'`

`Value` — Specifies strike type

string

Strike type, specified as a string. Choices are:

- `'single'` — Single horizontal line (default)
- `'none'` — No strikethrough line
- `'double'` — Double horizontal line

Examples

Create Paragraph with Double Strikethrough Text

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myStrikePresentation.pptx';
slides = Presentation(slidesFile);

titleSlide = add(slides,'Title and Content');
```

Create a paragraph and append text with 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);
```

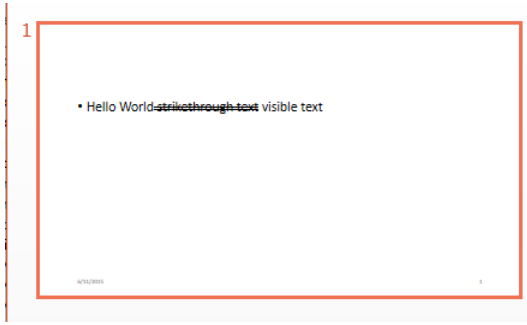
Add the paragraph to the slide and close the presentation.

```
replace(titleSlide,'Content',p);

close(slides);
```

Open `myStrikePresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```



- “Create and Format Text” on page 14-81

See Also

mlreportgen.ppt.Bold | mlreportgen.ppt.FontColor | mlreportgen.ppt.FontFamily | mlreportgen.ppt.FontSize | mlreportgen.ppt.Italic | mlreportgen.ppt.Underline

More About

- “Presentation Formatting Approaches” on page 14-20

Introduced in R2015b

mlreportgen.ppt.Subscript class

Package: mlreportgen.ppt

Subscript

Description

Subscript

Construction

`subscriptObj = Subscript()` creates a subscript object.

`subscriptObj = Subscript(value)` creates a subscript object.

Input Arguments

`value` — Display text as subscript

`[ ]` (default) | logical value

Display text as a subscript, specified as a logical. A setting of `true` (or `1`) renders text as a subscript.

Data Types: `logical`

Output Arguments

`subscriptObject` — Subscript

`mlreportgen.ppt.Subscript` object

Subscript, returned as an `mlreportgen.ppt.Subscript` object.

Properties

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

Examples

Use a Subscript

Set up a paragraph to display H_2O.

Set up a presentation.

```
import mlreportgen.ppt.*
slidesFile = 'mySubscript.pptx';
slides = Presentation(slidesFile);
slide1 = add(slides,'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');
```

Use the paragraph to replace the slide content, generate the presentation, and open the `mySubscript` presentation. (The `winopen` code works on Windows platforms.)

```
replace(slide1,'Content',para);
close(slides);

if ispc
```

```
    winopen(slidesFile);
end
```

- “Create and Format Text” on page 14-81

See Also

mlreportgen.ppt.Superscript

More About

- “Presentation Formatting Approaches” on page 14-20

Introduced in R2015b

mlreportgen.ppt.Superscript class

Package: mlreportgen.ppt

Superscript

Description

Superscript

Construction

`superscriptObj = Superscript()` creates a superscript object.

`superscriptObj = Superscript(value)` creates a superscript object.

Input Arguments

`value` — Display text as superscript
`[ ]` (default) | logical value

Display text as a superscript, specified as a logical. A setting of `true` (or `1`) renders text as a superscript.

Data Types: `logical`

Output Arguments

`superscriptObject` — Superscript
`mlreportgen.ppt.Superscript` object

Superscript, returned as an `mlreportgen.ppt.Superscript` object.

Properties

`Id` — ID for PPT API object
string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

Examples

Add a Superscript

Set up a paragraph to display x^2.

Set up a presentation.

```
import mlreportgen.ppt.*
slidesFile = 'mySuperscript.pptx';
slides = Presentation(slidesFile);
slide1 = add(slides,'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);
```

Use the paragraph to replace the slide content, generate the presentation, and open the `mySuperscript` presentation. (The `winopen` code works on Windows platforms.)

```
replace(slide1,'Content',para);
close(slides);

if ispc
    winopen(slidesFile);
```

```
end
```

- “Create and Format Text” on page 14-81

See Also

mlreportgen.ppt.Subscript

More About

- “Presentation Formatting Approaches” on page 14-20

Introduced in R2015b

mlreportgen.ppt.Table class

Package: mlreportgen.ppt

Table

Description

Table to include in a presentation.

Create a table using one of these approaches:

- Create an empty table and append table rows, with table entries for each column.
- Create a table, using an array or cell array to specify the table content.

After you create a table, you can add rows to it, and add entries to each table row. The row with the largest number of table entries determines the number of columns in the table if there are more table entries in the row than the number of rows specified in the `mlreportgen.ppt.Table` object constructor.

Construction

`tableObj = Table()` creates an empty table object.

`tableObj = Table(nCols)` creates an empty table object with the number of columns you specify.

`tableObj = Table(tableValues)` returns a table whose content is specified by a two-dimensional numeric array or a two-dimensional cell array of numbers, strings, or `mlreportgen.ppt.Paragraph` objects.

Input Arguments

`nCols` — Number of table columns

double

Number of table columns, specified as a double.

Data Types: `double`

`tableValues` — Table values

two-dimensional numeric array | two-dimensional cell array

Table values, specified as one of these:

- Two-dimensional numeric array
- Two-dimensional cell array of strings, numbers, or `mlreportgen.ppt.Paragraph` objects, or a combination of these kinds of values

Output Arguments

`tableObj` — Table

`mlreportgen.ppt.Table` object

Table, returned as an `mlreportgen.ppt.Table` object.

Properties

`NCols` — Number of table columns

double

Number of table columns, specified as a double.

Data Types: `double`

`Name` — Table name

string

Table name, specified as a string.

`StyleName` — Table style name

string

Table style name from the template for the presentation, specified as a string.

`X` — Upper-left x-coordinate position of table

string

Upper-left x-coordinate position of table, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the x-position units. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Y` — Upper-left y-coordinate position of table

string

Upper-left y-coordinate position of table, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the y-position units. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Width` — Width of table

string

Width of table, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the units for the width. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Height` — Height of table

string

Height of table, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the units for the height. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Style` — Table formatting

ignored

Table formatting. This property is ignored.

`Children` — Children of this PPT API object

cell array of `mlreportgen.ppt.Element` objects

This read-only property lists child elements that the object contains, specified as a cell array.

`Parent` — Parent of this PPT API object

PPT object

This read-only property lists the parent of this object, specified a PPT object.

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

Methods

Method	Purpose
`append`	Add content to table.
`replace`	Replace table.
`mlreportgen.ppt.Table.row`	Access table row.
`mlreportgen.ppt.Table.entry`	Access table entry.

Examples

Add a Table

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myTablePresentation.pptx';
```

```
slides = Presentation(slidesFile);

slide1 = add(slides,'Title and Table');
slide2 = add(slides,'Title and Table');
```

Create a table using a cell array.

```
table1 = Table({'a','b';'c','d'});
table1.Children(1).Style = {FontColor('red')};
table1.Children(2).Style = {FontColor('green')};
```

Use the `mlreportgen.ppt.Presentation.find` method to find the slides that have objects with a `Name` property set to `Table`. In the default PPT API, the `Title and Table` layout slide has a `Table` object.

```
contents = find(slides,'Table');
```

Replace the contents of the second slide with `table1`.

```
replace(contents(1),table1);
```

Create a second table using the MATLAB `magic` function.

```
table2 = Table(magic(9));
```

Replace the table in the third slide with `table2`.

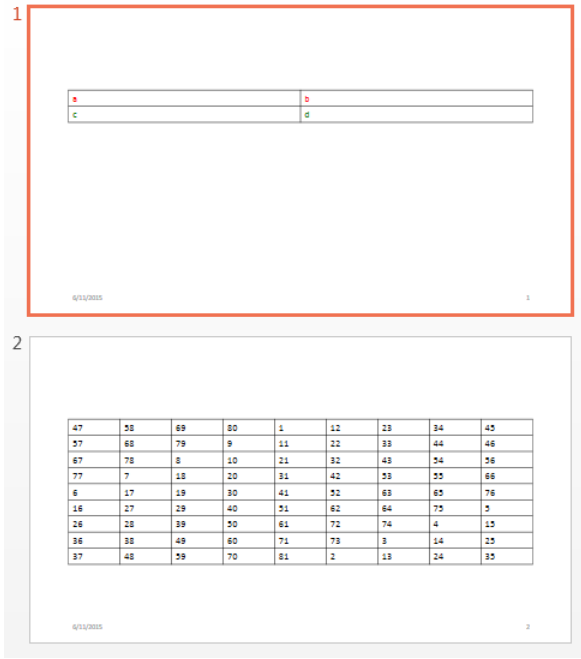
```
replace(contents(2),table2);
```

Close the presentation to generate the output.

```
close(slides);
```

Open `myTablePresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```



- “Create and Format Tables” on page 14-87

See Also

mlreportgen.ppt.TableEntry | mlreportgen.ppt.TablePlaceholder | mlreportgen.ppt.TableRow

Introduced in R2015b

mlreportgen.ppt.TableEntry class

Package: mlreportgen.ppt

Table entry

Description

Table entry to include in a table row.

To add content to a table entry, append a string or one or more `mlreportgen.ppt.Paragraph` objects.

The row with the largest number of table entries determines the number of columns in the table if there are more table entries in the row than the number of rows specified in the `mlreportgen.ppt.Table` object constructor.

Construction

`tableEntryObj = TableEntry()` creates an empty table entry object.

Output Arguments

`tableEntryObj` — Table entry
`mlreportgen.ppt.TableEntry` object

Table entry, returned as an `mlreportgen.ppt.TableEntry` object.

Properties

`Style` — Default formatting for text appended to table entry
cell array of PPT style objects

Default formatting for text appended to table entry, specified as an array of PPT style objects. You can include these `mlreportgen.ppt` style objects in the cell array:

- `BackgroundColor` object

- `FontFamily` object
- `FontSize` object
- `Bold` object
- `FontColor` object
- `Italic` object
- `Underline` object

`Children` — Children of this PPT API object

cell array of `mlreportgen.ppt.Element` objects

This read-only property lists child elements that the object contains, specified as a cell array.

`Parent` — Parent of this PPT API object

PPT object

This read-only property lists the parent of this object, specified a PPT object.

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

Methods

Method	Purpose
`append`	Add content to table entry.

Examples

Create a Table with Table Entries

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myTableEntryPresentation.pptx';
slides = Presentation(slidesFile);

slide1 = add(slides,'Title and Table');
```

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.Slide.find` method to find objects in the slide with the `Name` property set to `Table`. In the default PPT API template, the `Title and Table` layout slide has an object with the name `Table`.

```
contents = find(slide1,'Table');
```

Replace the table placeholder with `table1`.

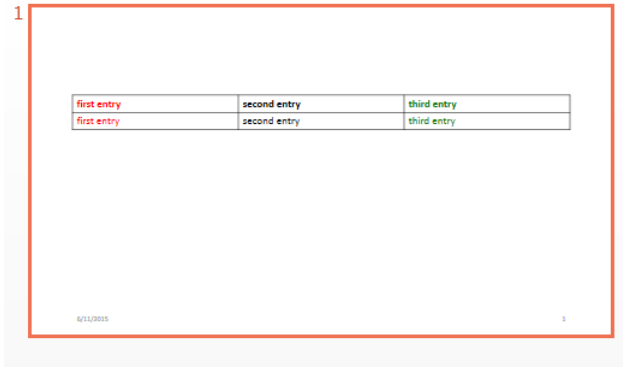
```
replace(contents(1),table1);
```

Close the presentation to generate the output.

```
close(slides);
```

Open `myTableEntryPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```



- “Create and Format Tables” on page 14-87

See Also

mlreportgen.ppt.Table | mlreportgen.ppt.Table.entry | mlreportgen.ppt.TableRow

Introduced in R2015b

mlreportgen.ppt.TablePlaceholder class

Package: mlreportgen.ppt

Slide placeholder to replace with table

Description

Slide placeholder to replace with a table. You can create a table placeholder using a layout slide. In the default PPT API, when you add a `Title and Table` slide to a presentation, the API creates a `TablePlaceholder` object. Use the `find` method with the slide object to find the picture placeholder. You can then set properties for the `TablePlaceholder` object.

Use the PowerPoint editor to insert a picture placeholder in a presentation:

1. Select the **Slide Master** tab.
2. Click the layout slide you want to add the picture placeholder to. You can add the placeholder to an existing or a new layout slide.
3. In the toolbar, click **Insert Placeholder** and select **Table**.

Properties

`Name` — Table placeholder name

string

Table placeholder name, specified as a string.

`X` — Upper-left x-coordinate position of placeholder

string

Upper-left x-coordinate position of placeholder, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the x-position units. Valid abbreviations are:

- No abbreviation — pixels

- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Y` — Upper-left y-coordinate position of placeholder

string

Upper-left y-coordinate position of placeholder, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the y-position units. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Width` — Width of placeholder

string

Width of placeholder, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the units for the width. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas

- `pt` — points
- `px` — pixels

`Height` — Height of placeholder

string

Height of placeholder, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the units for the height. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Style` — Table placeholder formatting

ignored

Table placeholder formatting. This property is ignored.

`Children` — Children of this PPT API object

cell array of `mlreportgen.ppt.Element` objects

This read-only property lists child elements that the object contains, specified as a cell array.

`Parent` — Parent of this PPT API object

PPT object

This read-only property lists the parent of this object, specified a PPT object.

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

Methods

Method	Purpose
`replace`	Replace table placeholder with table.

Examples

Replace Table Using a Table Placeholder

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myTablePlaceholderPresentation.pptx';
slides = Presentation(slidesFile);
slide1 = add(slides,'Title and Table');
```

Create a table.

```
table1 = Table(magic(9));
```

The `Title and Table` layout includes an object named `Table`. Use the `mlreportgen.ppt.Slide.find` method to find an object with the `Name` property of `Table`.

```
contents = find(slide1,'Table');
```

Replace the table placeholder with the table.

```
replace(contents(1),table1);
```

Close the presentation to generate the output.

```
close(slides);
```

Open `myTablePlaceholderPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```

- “Access PowerPoint Template Elements” on page 14-37
- “Add or Replace a Table” on page 14-78

See Also

mlreportgen.ppt.Table

Introduced in R2015b

mlreportgen.ppt.TableRow class

Package: mlreportgen.ppt

Table row

Description

Row to include in a table.

To add content to a table row, append `mlreportgen.ppt.TableEntry` objects to the row.

The row with the largest number of table entries determines the number of columns in the table if there are more table entries in the row than the number of rows specified in the `mlreportgen.ppt.Table` object constructor.

Construction

`tableRowObj = TableRow()` creates an empty table row object.

Output Arguments

`tableRowObj` — Table entry
`mlreportgen.ppt.TableRow` object

Table row, returned as an `mlreportgen.ppt.TableRow` object.

Properties

`Style` — Default formatting for text in table entries in row
cell array of PPT style objects

Default formatting for text in table entries in a table row, specified as an array of PPT style objects that specifies the format for the text. You can include these `mlreportgen.ppt` style objects in the cell array:

- `BackgroundColor` object
- `FontFamily` object
- `FontSize` object
- `Bold` object
- `FontColor` object
- `Italic` object
- `Underline` object

`Children` — Children of this PPT API object

cell array of `mlreportgen.ppt.Element` objects

This read-only property lists child elements that the object contains, specified as a cell array.

`Parent` — Parent of this PPT API object

PPT object

This read-only property lists the parent of this object, specified a PPT object.

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

Methods

Method	Purpose
`append`	Add content to table row.

Examples

Create a Table with Table Rows

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myTableEntryPresentation.pptx';
slides = Presentation(slidesFile);

slide1 = add(slides,'Title and Table');
```

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.Slide.find` method to find objects in the slide with the `Name` property set to `Table`. In the default PPT API template, the `Title and Table` layout slide has an object with the name `Table`.

```
contents = find(slide1,'Table');
```

Replace the table placeholder with `table1`.

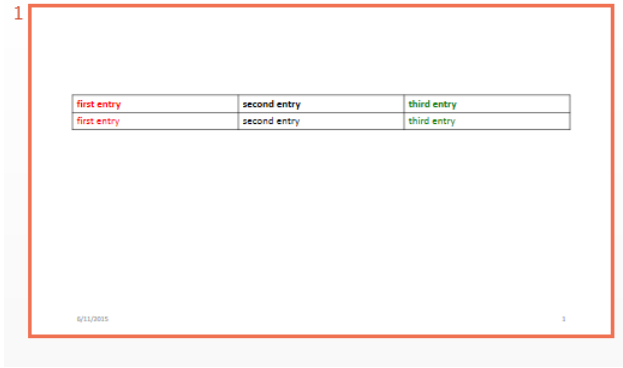
```
replace(contents(1),table1);
```

Close the presentation to generate the output.

```
close(slides);
```

Open `myTableEntryPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```



- “Create and Format Tables” on page 14-87

See Also

mlreportgen.ppt.Table.row | mlreportgen.ppt.Table | mlreportgen.ppt.TableEntry

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 PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier 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.*
slidesFile = 'myppt.pptx';
slides = Presentation(slidesFile);
add(slides,'Title and Content');

% Create the tableand 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(slides,'Content',table);
close(slides);

if ispc
winopen(slides.OutputPath);
end
```

- “Create and Format Tables” on page 14-87

See Also

mlreportgen.ppt.Table

Introduced in R2016a

mlreportgen.ppt.Text class

Package: mlreportgen.ppt

Text

Description

Text to include in a presentation.

Construction

`textObj = Text()` creates an empty text object.

`textObj = Text(text)` creates an `mlreportgen.ppt.Text` object with the text specified by `text`.

Input Arguments

`text` — Text
string

Text, specified as a string.

Output Arguments

`textObj` — Text
`mlreportgen.ppt.Text` object

Text, returned as an `mlreportgen.ppt.Text` object.

Properties

`Content` — Text content
string

Text content, specified as a string.

`Bold` — Option to use bold for text

logical value

Option to use bold for text, specified as a logical. To make text bold, set this property to `true` or `1`. Setting the `Bold` property adds a corresponding `mlreportgen.ppt.Bold` format object to the `Style` property of this presentation element. Removing the `Bold` property setting removes the object.

Data Types: `logical`

`FontColor` — Font color for presentation element

string

Font color, specified as a string. Use either a CSS color name or a hexadecimal RGB value.

- For a list of CSS color names, see http://www.crockford.com/wrrrld/color.html.
- To specify a hexadecimal RGB format, use `#` as the first character and two-digit hexidecimal 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`

`Subscript` — Format text as subscript

logical value

Format text as subscript, specifying as a logical value. A value of `true` renders the text as a subscript.

`Superscript` — Format text as superscript

logical value

Format text as superscript, specifying as a logical value. A value of `true` renders the text as a superscript.

`Underline` — Type of underline for text

[ ] (default) | string

Type of underlining for text, specified as a string. 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.

String	Description
`'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

`Style` — Text formatting

cell array of PPT style objects

Text formatting, specified as a cell array of PPT style objects. You can specify these `mlreportgen.ppt` style objects:

- `FontFamily` object

- `FontSize` object
- `Bold` object
- `FontColor` object
- `Italic` object
- `Underline` object

`Children` — Children of this PPT API object

cell array of `mlreportgen.ppt.Element` objects

This read-only property lists child elements that the object contains, specified as a cell array.

`Parent` — Parent of this PPT API object

PPT object

This read-only property lists the parent of this object, specified a PPT object.

`Tag` — Tag for this PPT API object

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

Examples

Use Text Objects in a Presentation

Create a presentation with two slides.

```
import mlreportgen.ppt.*

slidesFile = 'myTextPresentation.pptx';
slides = Presentation(slidesFile);

add(slides,'Title Slide');
add(slides,'Title and Content');
```

Create a `Paragraph` object to use for the title of the presentation. Make the text bold and red.

```
p = Paragraph();
p.Bold = true;
p.FontColor = 'red';
```

Create a `Text` object and append it to the paragraph.

```
titleText = Text('My Presentation Title');
append(p,titleText);
```

Replace the title for the first slide with the paragraph.

```
contents = find(slides,'Title');
replace(contents(1),p);
```

Create a paragraph for the content of the second slide.

```
p1 = Paragraph('My content');
append(p1,Text(' for the second slide'));
```

Replace the content with the `p1` paragraph.

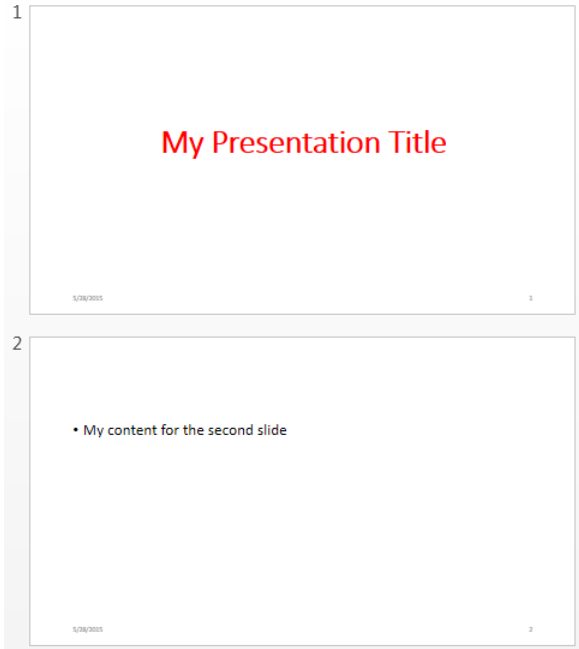
```
replace(slides,'Content',p1);
```

Close the presentation.

```
close(slides);
```

Open `myParagraphPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```



- “Create and Format Text” on page 14-81

See Also

mlreportgen.ppt.Paragraph | mlreportgen.ppt.TextBox | mlreportgen.ppt.TextBoxPlaceholder

mlreportgen.ppt.TextBox class

Package: mlreportgen.ppt

Text box

Description

Text box to include in a presentation.

Construction

`textBoxObj = TextBox()` creates an empty text box object.

Output Arguments

`textBoxObj` — Text box
`mlreportgen.ppt.TextBox` object

Text box, returned as an `mlreportgen.ppt.TextBox` object.

Properties

`Bold` — Option to use bold for text
logical value

Option to use bold for text, specified as a logical. To make text bold, set this property to `true` or `1`. Setting the `Bold` property adds a corresponding `mlreportgen.ppt.Bold` format object to the `Style` property of this presentation element. Removing the `Bold` property setting removes the object.

Data Types: `logical`

`FontColor` — Font color for presentation element
string

Font color, specified as a string. Use either a CSS color name or a hexadecimal RGB value.

- For a list of CSS color names, see http://www.crockford.com/wrrrld/color.html.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexidecimal 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) | string

Type of underlining for text, specified as a string. 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.

String	Description
`'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

String	Description
`'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

string

Text box name, specified as a string.

`X` — Upper-left x-coordinate position of text box

string

Upper-left x-coordinate of text box, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the x-position units. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Y` — Upper-left y-coordinate position of text box

string

Upper —left y-coordinate position of text box, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the y-position units. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Width` — Width of text box

string

Width of text box, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the units for the width. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Height` — Height of text box

string

Height of text box, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the units for the height. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas

- `pt` — points
- `px` — pixels

`Style` — Text formatting

cell array of PPT style objects

Text formatting, specified as a cell array of PPT style objects. You can specify these `mlreportgen.ppt` style 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 `mlreportgen.ppt.Element` objects

This read-only property lists child elements that the object contains, specified as a cell array.

`Parent` — Parent of this PPT API object

PPT object

This read-only property lists the parent of this object, specified a PPT object.

`Tag` — Tag for this PPT API object

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

Methods

Method	Purpose
`add`	Add content to text box.
`replace`	Replace text box content.

Examples

Add a Text Box

Create a presentation with two slides.

```
import mlreportgen.ppt.*
slides = Presentation('myTextBoxPresentation.pptx');
```

Add a blank slide.

```
blank = add(slides,'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.

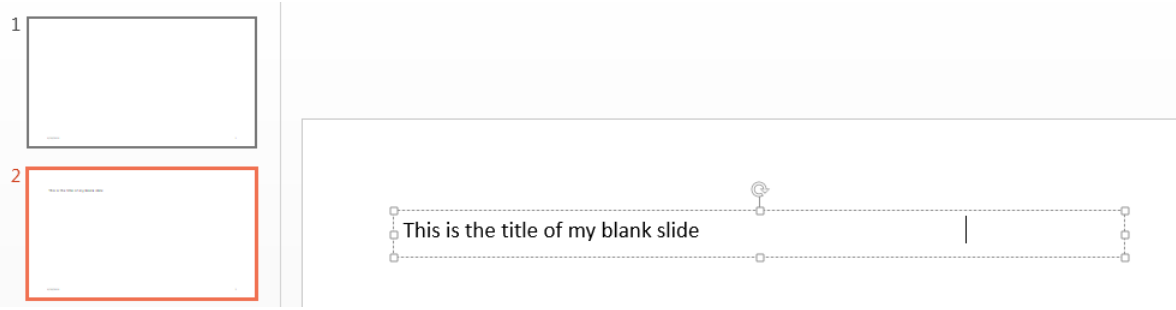
```
add(tb,'This is the title of my blank slide');
add(blank,tb);
```

Close the presentation.

```
close(slides);
```

Open `myTextBoxPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```



- “Create and Format Text” on page 14-81

See Also

mlreportgen.ppt.Paragraph | mlreportgen.ppt.Text | mlreportgen.ppt.TextBoxPlaceholder

mlreportgen.ppt.TextBoxPlaceholder class

Package: mlreportgen.ppt

Slide placeholder to replace with text box

Description

Slide placeholder to replace with a text box. You can create a text box placeholder using a layout slide. In the default PPT API, when you add a `Title Slide` slide to a presentation, the API creates a `TextBoxPlaceholder` object. Use the `find` method with the slide object, to find the text box placeholder. You can then set properties for that `TextBoxPlaceholder` object.

To use the PowerPoint editor to insert a picture placeholder in a presentation:

1. Select the **Slide Master** tab.
2. Click the layout slide you want to add the picture placeholder to. You can add the placeholder to an existing or a new layout slide.
3. In the toolbar, click **Insert Placeholder** and select **Text**.

Properties

`Bold` — Option to use bold for text

logical value

Option to use bold for text, specified as a logical. To make text bold, set this property to `true` or `1`. Setting the `Bold` property adds a corresponding `mlreportgen.ppt.Bold` format object to the `Style` property of this presentation element. Removing the `Bold` property setting removes the object.

Data Types: `logical`

`FontColor` — Font color for presentation element

string

Font color, specified as a string. Use either a CSS color name or a hexadecimal RGB value.

- For a list of CSS color names, see http://www.crockford.com/wrrrld/color.html.
- To specify a hexadecimal RGB format, use # as the first character and two-digit hexidecimal 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) | string

Type of underlining for text, specified as a string. 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.

String	Description
`'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

String	Description
`'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 placeholder name

string

Text box placeholder name, specified as a string.

`X` — Upper-left x-coordinate position of placeholder

string

Upper-left x-coordinate position of placeholder, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the x-position units. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Y` — Upper-left y-coordinate position of placeholder

string

Upper-left y-coordinate position of placeholder, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the y-position units. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters

- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Width` — Width of placeholder

string

Width of placeholder, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the units for the width. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points
- `px` — pixels

`Height` — Height of placeholder

string

Height of placeholder, specified as a string.

The string must have the format `valueUnits` where `Units` is an abbreviation for the units for the height. Valid abbreviations are:

- No abbreviation — pixels
- `cm` — centimeters
- `in` — inches
- `mm` — millimeters
- `pi` — picas
- `pt` — points

- `px` — pixels

`Style` — Text formatting

cell array of PPT style objects

Text formatting, specified as an array of PPT style objects. You can specify these `mlreportgen.ppt` style objects:

- `BackgroundColor` object
- `FontFamily` object
- `FontSize` object
- `FontColor` object
- `Italic` object
- `Underline` object

`Children` — Children of this PPT API object

cell array of `mlreportgen.ppt.Element` objects

This read-only property lists child elements that the object contains, specified as a cell array.

`Parent` — Parent of this PPT API object

PPT object

This read-only property lists the parent of this object, specified a PPT object.

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

Methods

Method	Purpose
`add`	Add content to placeholder.
`replace`	Replace placeholder content.

Examples

Replace Text Using a Text Box Placeholder

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myTextBoxPlaceholderPresentation.pptx';
slides = Presentation(slidesFile);
slide1 = add(slides,'Title Slide');
```

Create a paragraph.

```
p = Paragraph('My Presentation Title');
```

The `Title Slide` layout includes a text box object named `Title`. Use the `mlreportgen.ppt.Slide.find` method to find an object with the `Name` property of `Title`.

```
contents = find(slide1,'Title')
```

```
contents =

  TextBoxPlaceholder with properties:

          Bold: []
     FontColor: []
        Italic: []
        Strike: []
     Subscript: []
   Superscript: []
     Underline: []
          Name: 'Title'
             X: []
```

```
             Y: []
         Width: []
        Height: []
         Style: []
      Children: []
        Parent: [1x1 mlreportgen.ppt.Slide]
           Tag: 'ppt.TextBoxPlaceholder:5417:755'
            Id: '5417:755'
```

Replace the placeholder with the paragraph.

```
replace(contents(1),p);
```

Close the presentation.

```
close(slides);
```

Open `myTextBoxPlaceholderPresentation.pptx`. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```

- “Access PowerPoint Template Elements” on page 14-37
- “Add and Replace Text” on page 14-75

See Also

mlreportgen.ppt.TextBox

Introduced in R2015b

mlreportgen.ppt.Underline class

Package: mlreportgen.ppt

Underline text

Description

Underline text.

Construction

`underline = Underline()` draws a single line under text.

`underline = Underline(type)` draws a line of the specified type under the text.

Input Arguments

`type` — Style of underline

string

Underline style, specified as a string. Valid strings are:

String	Description
`'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

String	Description
`'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

Output Arguments

`underline` — Underline text

`mlreportgen.ppt.Underline` object

Underline, returned as an `mlreportgen.ppt.Underline` object.

Properties

`Type` — Underline style

string

Underline style, specified as a string. See the description of the `type` input argument for the constructor.

Data Types: `char`

`Id` — ID for PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

Examples

Create Underlined Text

Create a presentation.

```
import mlreportgen.ppt.*

slidesFile = 'myUnderlinePresentation.pptx';
slides = Presentation(slidesFile);

titleSlide = add(slides,'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 and close the presentation.

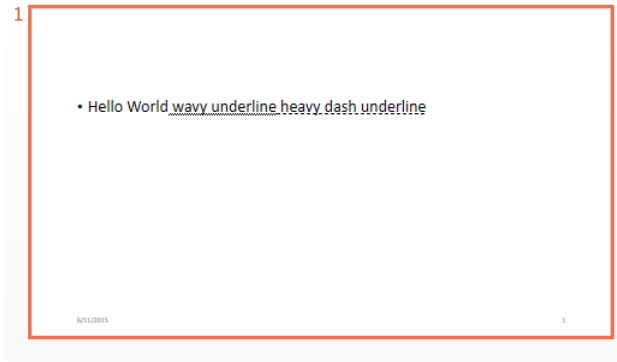
```
replace(titleSlide,'Content',p);

close(slides);
```

Open the `myUnderlinePresentation` presentation. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
```

```
end
```



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) | string

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 PPT API object

string

ID for PPT API object, specified as a string. A session-unique ID is generated as part of object creation. You can specify an ID to replace the generated ID.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Value` — Vertical alignment of table entry content

`'top'` (default) | `'bottom'` | `'middle'` | `'topCentered'` | `'middleCentered'` | `'bottomCentered'`

Vertical alignment of table entry content, specified as a string.

Examples

Add a Table

Create a presentation.

```
import mlreportgen.ppt.*
slidesFile = 'myVAlign.pptx';
```

```
slides = Presentation(slidesFile);
slide1 = add(slides,'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);

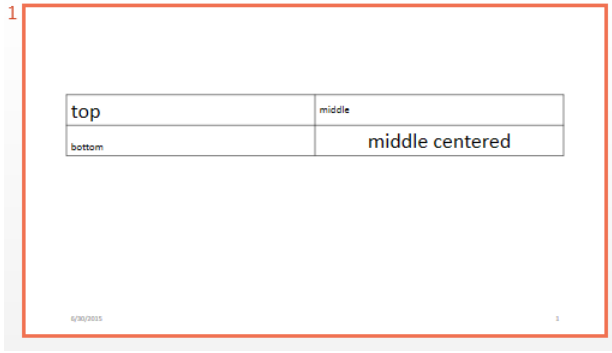
p4 = Paragraph('middle centered');
r2e2 = TableEntry();
r2e2.Style = {VAlign('middleCentered'),FontSize('.5in')};
append(r2e2,p4);
append(row2,r2e2);

append(table1,row1);
append(table1,row2);
```

Add the table to the slide, generate the presentation, and open the `myVAlign` presentation. (The `winopen` code works on Windows platforms.)

```
replace(slide1,'Content',table1);
close(slides);

if ispc
    winopen(slidesFile);
end
```



- “Create and Format Tables” on page 14-87

See Also

mlreportgen.ppt.HAlign | mlreportgen.ppt.TableEntry

More About

- “Presentation Formatting Approaches” on page 14-20

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
string

The text to display for the message, specified as a string.

`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 PPT API object

string

ID for PPT API object, specified as a string. 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

session-unique tag generated as part of object creation (default) | string

Tag for this PPT API object, specified as a string. 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.

An example of a reason for specifying your own tag value is to make it easier to identify where an issue occurred during presentation generation.

`Text` — Text of message

string

Message text, specified as a string.

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);
```

- “Display Presentation Generation Messages” on page 14-15

See Also

`mlreportgen.ppt.MessageDispatcher.dispatch` | mlreportgen.ppt.MessageEventData

Introduced in R2015b

Create a Report Program

- "Create a Report Program" on page 13-3
- "Document Object Model" on page 13-4
- "Construct a DOM Object" on page 13-5
- "Import the DOM API Package" on page 13-6
- "Use Dot Notation for DOM Object Properties" on page 13-7
- "Create a Document Object to Hold Content" on page 13-9
- "Add Content to a Report" on page 13-11
- "Clone a DOM Object" on page 13-13
- "Add Content as a Group" on page 13-14
- "Stream a Report" on page 13-16
- "Report Packages" on page 13-17
- "Close a Report" on page 13-18
- "Display a Report" on page 13-19
- "Report Formatting Approaches" on page 13-21
- "Use Style Sheet Styles" on page 13-24
- "Format Inheritance" on page 13-26
- "Templates for DOM API Report Programs" on page 13-27
- "Create Object Containers" on page 13-30
- "Form-Based Reporting" on page 13-32
- "Fill the Blanks in a Report Form" on page 13-33
- "Use Subforms in a Report" on page 13-35
- "Create a Microsoft Word Document Part Template Library" on page 13-37
- "Create an HTML Document Part Template Library" on page 13-40
- "Create a PDF Document Part Template Library" on page 13-42
- "Object-Oriented Report Creation" on page 13-48

- "Simplify Filling in Forms" on page 13-49
- "Create and Format Text" on page 13-51
- "Create and Format Paragraphs" on page 13-57
- "Create and Format Lists" on page 13-63
- "Create and Format Tables" on page 13-69
- "Create Links" on page 13-83
- "Create and Format Images" on page 13-86
- "Create a Table of Contents" on page 13-88
- "Create Image Maps" on page 13-96
- "Automatically Number Document Content" on page 13-98
- "Appending HTML to DOM Reports" on page 13-103
- "Append HTML Content to DOM Reports" on page 13-105
- "Append HTML File Contents to DOM Reports" on page 13-107
- "Use an HTML Cleanup Program" on page 13-109
- "HTML Code Requirements for DOM Reports" on page 13-113
- "Display Report Generation Messages" on page 13-119
- "Compile a Report Program" on page 13-123
- "Create a Microsoft Word Template" on page 13-124
- "Add Holes in a Microsoft Word Template" on page 13-125
- "Modify Styles in a Microsoft Word Template" on page 13-130
- "Create an HTML or PDF Template" on page 13-135
- "Add Holes in HTML and PDF Templates" on page 13-137
- "PDF and HTML Document Parts and Holes" on page 13-139
- "Modify Styles in HTML Templates" on page 13-143
- "Modify Styles in PDF Templates" on page 13-144
- "Create Page Layout Sections" on page 13-148
- "Create Page Footers and Headers" on page 13-152
- "Add Complex Page Numbers in Microsoft Word" on page 13-162

Create a Report Program

The MATLAB Report Generator includes a set of functions, called the document object model (DOM) API, that allows you to generate Word, HTML, and PDF reports programmatically. For example, this MATLAB code uses the API to generate and display an HTML report that shows today's date.

```
import mlreportgen.dom.*;
report = Document('today');
append(report, ['Today is ', date, '.']);
close(report);
rptview(report.OutputPath);
```

To get started learning about creating reports with the DOM API, see "Document Object Model" on page 13-4.

More About

- "Document Object Model" on page 13-4

Document Object Model

The DOM API creates a representation of a report document in your system's memory. Such a representation is often referred to as a document object model (DOM).

The DOM API's document object model consists of a hierarchical set of data structures, known as objects, that represent the document and its contents. At the top of the hierarchy is an object representing the document. The document object maintains a list of objects, called its children, that represent its contents (such as paragraphs, images, tables, and lists). Each child object, in turn, maintains a list of its contents. For example, a table lists its rows, a row lists its table entries, and a table entry lists its contents.

The DOM API has functions that allow you to create and assemble objects, such as paragraphs, images, and tables, into a model of a specific document. You can then use the API to write the model out to disk as an HTML, Microsoft Word, or PDF file.

DOM Object Help and Documentation

For a list of the DOM objects, enter this command at the MATLAB prompt.

```
help mlreportgen.dom
```

To get help for a specific object or method, use a `help` command.

```
help mlreportgen.dom.Paragraph
```

For a complete list of DOM API classes and functions in the MATLAB Report Generator documentation, open the **Functions and Other Reference** page.

To see the documentation reference page for an object, search in documentation or in MATLAB use a `doc` command.

```
doc mlreportgen.dom.Paragraph
```

Related Examples

- "Construct a DOM Object" on page 13-5
- "Use Dot Notation for DOM Object Properties" on page 13-7
- "Import the DOM API Package" on page 13-6

Construct a DOM Object

The DOM API includes a set of MATLAB functions, called constructors, for creating DOM objects of various types, or classes.

The name of an object constructor is the name of the MATLAB class from which the DOM creates an object. 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`.

```
p = 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(p,'In the Beginning');
```

Related Examples

- “Import the DOM API Package” on page 13-6
- “Use Dot Notation for DOM Object Properties” on page 13-7

More About

- “Document Object Model” on page 13-4

Import the DOM API Package

All DOM class names, including constructor names, include the prefix `mlreportgen.dom`. You can omit the prefix if you insert this statement at the beginning of any program or function that uses the DOM API.

```
import mlreportgen.dom.*;
```

Examples that refer to DOM API objects and functions without the `mlreportgen.dom` prefix assume that you have already imported the DOM API package.

Related Examples

- “Create a Report Program” on page 13-3

More About

- “Document Object Model” on page 13-4

Use Dot Notation for DOM Object Properties

One way to specify the property of a document object is to use dot notation. You can specify a property this way to get the current value or to set the value.

With dot notation, you append a period to the name of a variable that references the object and then add the property name. For example, a `Document` object has an `OutputPath` property that specifies the location of the report. This code assigns the document object to the variable `d`. After you generate the report, you can use the value that `d.OutputPath` returns as input to the `rptview` command.

```
import mlreportgen.dom.*;
doctype = 'pdf';
d = Document('mydoc',doctype);

p = Paragraph('Hello World');
append(d,p);

close(d);
rptview(d.OutputPath);
```

This code sets a property using dot notation. Create a `Paragraph` object and assign it to the variable `p`. Use dot notation with the variable to set the `Style` property. The `Style` property can use a cell array as its value, enabling you to set several format properties at once.

```
import mlreportgen.dom.*;
doctype = 'docx';
d = Document('mydoc',doctype);

p = Paragraph('Hello World');
p.Style = {FontSize('11pt'),FontFamily('Arial'),Underline('single')};
append(d,p);

close(d);
rptview(d.OutputPath);
```

Related Examples

- “Construct a DOM Object” on page 13-5
- “Format Properties” on page 13-22

More About

- “Document Object Model” on page 13-4

Create a Document Object to Hold Content

Every report program must create an `mlreportgen.dom.Document` object to hold report content. Use the `mlreportgen.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

Functions

`rptview`

Classes

mlreportgen.dom.Document

Related Examples

- “Create a Report Program” on page 13-3
- “Templates for DOM API Report Programs” on page 13-27
- “Construct a DOM Object” on page 13-5

More About

- “Form-Based Reporting” on page 13-32
- “Document Object Model” on page 13-4

Add Content to a Report

The DOM `append` method allows you to add content to documents, paragraphs, tables, and other DOM objects that serve as containers for report content. The `append` function takes two arguments. The first argument is the object to append the content to. The second is the content to append. This example appends the text `Hello World` to the document.

```
d = Document('MyReport');
append(d,'Hello World');
```

If you cannot append the second object to the first, the `append` function returns an error. For example, the `append` method in this code returns an error because you cannot append a paragraph to an image.

```
% This code returns an error
image = Image('membrane.png');
append(image,Paragraph('Hello World'));
```

The reference documentation for each class lists the types of objects that you can append to instances.

Depending on the target object type, the `append` method allows you to append strings, doubles, arrays, and other basic MATLAB data types directly. The method converts the data to a DOM object before appending it to the target object. For example, this code appends a two-dimensional array of strings to a document as a table.

```
d = Document('MyDoc');
tableArray = {'a','b';'c','d'};
append(d,tableArray);
```

You can also specify basic MATLAB data types as initial content for many constructors. This example creates a table object that has a two-dimensional array of strings as initial content.

```
d = Document('MyDoc');
tableArray = {'a','b';'c','d'};
append(d,Table(tableArray));
```

See Also

Functions

mlreportgen.dom.Paragraph.append

Related Examples

- “Construct a DOM Object” on page 13-5
- “Clone a DOM Object” on page 13-13
- “Add Content as a Group” on page 13-14
- “Stream a Report” on page 13-16

More About

- “Document Object Model” on page 13-4

Clone a DOM Object

You can append the same object once in a program. To append an object multiple times to one object or to multiple objects, use the `clone` function, which creates copies of the object.

```
import mlreportgen.dom.*;

d = Document('MyDoc');
text = Text('Hello World');
text.Color = 'magenta';
text2 = clone(text);
text2.Color = 'cyan';
append(d,text);
append(d,text2);

close(d);
rptview(d.OutputPath);
```

See Also

Functions

mlreportgen.dom.Paragraph.clone

Related Examples

- “Add Content to a Report” on page 13-11
- “Construct a DOM Object” on page 13-5

More About

- “Document Object Model” on page 13-4

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

mlreportgen.dom.Paragraph.append

Classes

mlreportgen.dom.Group

Related Examples

- “Add Content to a Report” on page 13-11

Stream a Report

The DOM API supports two modes of appending content to a document:

- In-memory — Creates the document entirely in memory. In-memory is the default mode.
- Streaming — Streaming mode writes objects to disk as they are appended to a document. Streaming mode allows you to create large reports on systems with modest memory.

To enable streaming mode, set the `StreamOutput` property of the `Document` object for the report to `true`.

```
d = Document('MyDoc');
d.StreamOutput = true;
```

See Also

Classes

mlreportgen.dom.Document

Related Examples

- “Add Content to a Report” on page 13-11

Report 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 `htmx` compressed file extension. By default, the DOM API generates HTML reports as `.htmtx` 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

`unzipTemplate` | `zipTemplate`

Classes

mlreportgen.dom.Document

More About

- “Document Object Model” on page 13-4

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

Functions

mlreportgen.dom.Document.close

Related Examples

- “Create a Report Program” on page 13-3

More About

- “Document Object Model” on page 13-4

Display a Report

The DOM API `rptview` function allows 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 any 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 13-27.

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 13-24.

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.

```
p = Paragraph('Danger!');
p.Style = {Color('red'),FontFamily('Arial'),FontSize('18pt')};
```

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:

```
warning = {Color('red'),FontFamily('Arial'),FontSize('18pt')};
```

```
p = Paragraph('Danger!');
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. You can use dot notation to specify formats for an object. 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. Each string after the dot corresponds to a format property.

```
p = Paragraph('Danger!');
p.Color = 'red';
p.FontFamilyName = 'Arial';
p.FontSize = '18pt';
```

Assigning a value to a format property causes the API to create an equivalent format object and assign it to the `Style` property of the document object. Similarly, assigning a format object to an object's `Style` property causes the API to assign an equivalent value to the corresponding format property, if it exists. In this way, the API keeps format properties for an object synchronized with the `Style` property of the object.

Note: When you change the value of a format property, the DOM API:

- Creates a clone of the corresponding format object
- Changes the value of the clone's corresponding format object property
- Replaces the original format object with the clone in the array of format objects assigned to the document object

In this way, the DOM prevents a change in a format property in one object from changing a style originally assigned to other objects.

Related Examples

- "Use Style Sheet Styles" on page 13-24

More About

- "Format Inheritance" on page 13-26

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 13-144.

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 string.

```
p = Paragraph('Use care when unplugging this device','Warning');
```

Related Examples

- "Create a Microsoft Word Template" on page 13-124
- "Modify Styles in a Microsoft Word Template" on page 13-130
- "Create an HTML or PDF Template" on page 13-135
- "Modify Styles in HTML Templates" on page 13-143
- "Modify Styles in PDF Templates" on page 13-144

More About

- "Report Formatting Approaches" on page 13-21
- "Format Inheritance" on page 13-26

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.

```
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.

```
d = Document('MyDoc');
tableArray = {'a','b';'c','d'};
table = append(d,tableArray);
table.Style = {Color('blue')};
append(d,table);
close(d);
```

Related Examples

- “Use Style Sheet Styles” on page 13-24

More About

- “Report Formatting Approaches” on page 13-21

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 13-32.

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). 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 `.htmtx` 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 13-24.

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 13-130.

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 13-143 and "Modify Styles in PDF Templates" on page 13-144

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 13-148.

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 13-37.

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 13-40 and “Create a PDF Document Part Template Library” on page 13-42.

See Also

mlreportgen.dom.Document.createTemplate | `unzipTemplate` | `zipTemplate`

Related Examples

- “Use Style Sheet Styles” on page 13-24
- “Modify Styles in PDF Templates” on page 13-144
- “Modify Styles in HTML Templates” on page 13-143
- “Create a Microsoft Word Document Part Template Library” on page 13-37
- “Create an HTML Document Part Template Library” on page 13-40
- “Create a PDF Document Part Template Library” on page 13-42

Create Object Containers

You can use an `mlreportgen.dom.Container` object to create an HTML container object, such as a `div`, `section`, or `article`, not otherwise supported by the DOM API and to simulate HTML format inheritance in Word output.

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 ensure 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. It applies the formats specified by the `Container` object `Style` attribute to each child, unless overridden by the child. It then appends the child to the output. Word and PDF output ignore the `HTMLTag` and `StyleName` properties of the `Container` object.

Tip To produce collections of document elements, you can use `mlreportgen.dom.Container` or `mlreportgen.dom.Group` objects.

- 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.

See Also

Classes

mlreportgen.dom.Container | mlreportgen.dom.Group

Related Examples

- "Use Style Sheet Styles" on page 13-24

More About

- "Report Formatting Approaches" on page 13-21

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.

Related Examples

- "Fill the Blanks in a Report Form" on page 13-33
- "Use Subforms in a Report" on page 13-35
- "Create a Microsoft Word Template" on page 13-124
- "Add Holes in a Microsoft Word Template" on page 13-125
- "Create an HTML or PDF Template" on page 13-135
- "Add Holes in HTML and PDF Templates" on page 13-137

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 strings.

```
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
mlreportgen.dom.Document.moveToNextHole

Related Examples

- “Use Subforms in a Report” on page 13-35
- “Create a Microsoft Word Template” on page 13-124
- “Add Holes in a Microsoft Word Template” on page 13-125
- “Create an HTML or PDF Template” on page 13-135
- “Add Holes in HTML and PDF Templates” on page 13-137

More About

- “Form-Based Reporting” on page 13-32

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 `mlreportgen.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 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 13-37.

See Also

Functions

mlreportgen.dom.Document.moveToNextHole

Classes

mlreportgen.dom.DocumentPart

Related Examples

- "Fill the Blanks in a Report Form" on page 13-33
- "Create a Microsoft Word Template" on page 13-124
- "Add Holes in a Microsoft Word Template" on page 13-125
- "Create an HTML or PDF Template" on page 13-135
- "Add Holes in HTML and PDF Templates" on page 13-137

More About

- "Form-Based Reporting" on page 13-32

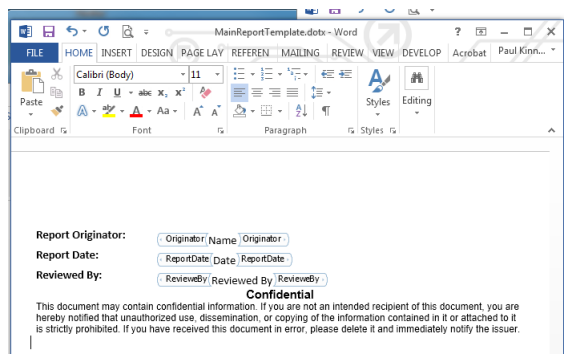
Create a Microsoft Word Document Part Template Library

A document part template library is a set of document part templates stored by name in a template file. 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. 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.

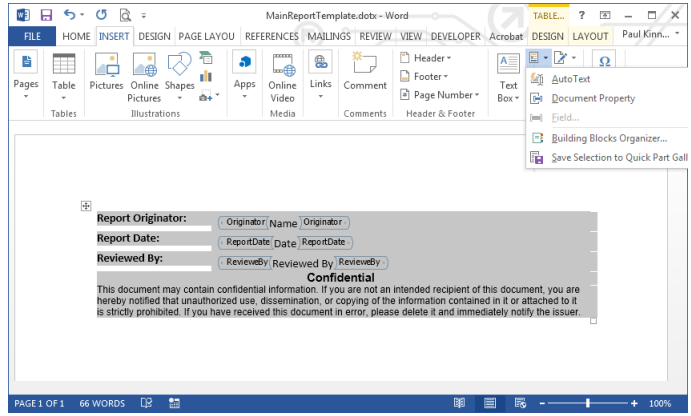
Create Document Part Template Library in a Word Template

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.

See Also

Classes

mlreportgen.dom.DocumentPart

Related Examples

- "Fill the Blanks in a Report Form" on page 13-33
- "Create a Microsoft Word Template" on page 13-124
- "Add Holes in a Microsoft Word Template" on page 13-125
- "Add Holes in HTML and PDF Templates" on page 13-137
- "Create a PDF Document Part Template Library" on page 13-42
- "Create an HTML Document Part Template Library" on page 13-40

More About

- "Form-Based Reporting" on page 13-32

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 13-88.

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 13-40.
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 13-143.
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

`unzipTemplate` | `zipTemplate`

Related Examples

- “Create an HTML or PDF Template” on page 13-135
- “Add Holes in HTML and PDF Templates” on page 13-137
- “Create a PDF Document Part Template Library” on page 13-42
- “Create a Microsoft Word Document Part Template Library” on page 13-37

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 13-42.
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 13-144.
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 13-88
- “Automatically Number Document Content” on page 13-98
- “Add Holes in HTML and PDF Templates” on page 13-137
- “Create Page Layout Sections” on page 13-148
- “Create Page Footers and Headers” on page 13-152
- “PDF and HTML Document Parts and Holes” on page 13-139
- “Create a Page Reference” on page 13-84

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 bottom left right header footer gutter; page-size: height width orientation`
		`first-page-number`	Number of first page in the layout

Purpose	Element	Attributes	Values
		`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`	String that identifies hole by name
		`default-style-name`	Style sheet style to use when style is not set programmatically
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	

Purpose	Element	Attributes	Values
Page break	`pagebreak`	No attributes	
Numeric reference to page where target is located	`pageref`	`target`	ID of target; create target in your report using mlreportgen.dom.LinkTarget
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

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

Standard HTML Elements

You can use these standard HTML elements in PDF templates.

HTML Element	Attributes
`a`	`class`, `style`, `href`, `name`
`b`	`class`, `style`
`body`	`class`, `style`
`br`	n/a

HTML Element	Attributes
code	class, style
del	class, style
div	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, alt
li	class, style
ol	class, style
p	class, style, align
pre	class, style
s	class, style
span	class, style
strike	class, style
sub	class, style
sup	class, style
table	class, style, align, bgcolor, border, cellspacing, cellpadding, 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, valign, nowrap
tr	class, style, bgcolor, valign
tt	class, style

HTML Element	Attributes
`u`	`class`, `style`
`ul`	`class`, `style`

For information about these elements, see the W3Schools tags documentation at www.w3schools.com/tags.

See Also

`unzipTemplate` | `zipTemplate`

Related Examples

- “Create an HTML or PDF Template” on page 13-135
- “Add Holes in HTML and PDF Templates” on page 13-137
- “Create an HTML Document Part Template Library” on page 13-40
- “Create a Microsoft Word Document Part Template Library” on page 13-37

Object-Oriented Report Creation

Note: For information on object-oriented programming in MATLAB, see "Object-Oriented Programming".

The DOM API supports an object-oriented approach to creating report programs. With this approach, you subclass the DOM `Document` and `DocumentPart` classes to create document and document part classes tailored to your report application. You then create instances of these classes to generate a report.

Related Examples

- "Simplify Filling in Forms" on page 13-49

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 example.

See Also

Functions

mlreportgen.dom.Document.moveToNextHole

Classes

mlreportgen.dom.DocumentPart

Related Examples

- “Use Subforms in a Report” on page 13-35
- “Fill the Blanks in a Report Form” on page 13-33
- “Create a Microsoft Word Template” on page 13-124
- “Add Holes in a Microsoft Word Template” on page 13-125
- “Create an HTML or PDF Template” on page 13-135
- “Add Holes in HTML and PDF Templates” on page 13-137

More About

- “Form-Based Reporting” on page 13-32

Create and Format Text

In this section...
“Create Text” on page 13-51
“Create Special Characters” on page 13-51
“Append HTML or XML Markup” on page 13-52
“Format Text” on page 13-52

Create Text

You can create text by appending a string to a document, paragraph, table entry, or list item. The DOM `append` function converts the string 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 the string `'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 http://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 13-21.

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
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 13-130.

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 13-144.

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.Bold | mlreportgen.dom.CharEntity | mlreportgen.dom.FontFamily | mlreportgen.dom.FontSize | mlreportgen.dom.Italic | mlreportgen.dom.Strike | mlreportgen.dom.Text | mlreportgen.dom.Underline

Related Examples

- “Add Content to a Report” on page 13-11
- “Modify Styles in HTML Templates” on page 13-143

More About

- "Report Formatting Approaches" on page 13-21

Create and Format Paragraphs

In this section...
“Create a Paragraph” on page 13-57
“Create a Heading” on page 13-57
“Format a Paragraph” on page 13-57

Create a Paragraph

You can create a paragraph by using an `mlreportgen.dom.Paragraph` constructor with a text string. 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 a `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 13-21.

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`

Formatting	Format Object	Format Property
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
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 13-130.

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 13-144.

For more information about using HTML styles with DOM objects, see "Modify Styles in HTML Templates" on page 13-143.

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.Bold | mlreportgen.dom.Display | mlreportgen.dom.FontFamily | mlreportgen.dom.FontSize | mlreportgen.dom.Italic | mlreportgen.dom.KeepLinesTogether | mlreportgen.dom.KeepWithNext | mlreportgen.dom.LineSpacing | mlreportgen.dom.PageBreakBefore | mlreportgen.dom.Paragraph | mlreportgen.dom.Strike | mlreportgen.dom.Text | mlreportgen.dom.Underline

Related Examples

- “Add Content to a Report” on page 13-11

More About

- “Report Formatting Approaches” on page 13-21

Create and Format Lists

In this section...
“Create an Ordered or Unordered List” on page 13-63
“Create a Multilevel List” on page 13-65
“Format Lists” on page 13-66

You can add two kinds of lists to a report:

- Unordered (bulleted)
- Ordered (numbered)
- Multilevel (lists that contain ordered or unordered lists in any combination)

Create an Ordered or Unordered List

You can create lists from a numeric or cell array or one item at a time.

- Creating a list from a cell array allows you to include items of different types in the list.
- Creating a list one item at a time is useful for including multiple objects in a list item.

Create an Unordered List from an Array

You can create an unordered list by appending a one-dimensional numeric or cell array to a document (or document part). The `append` function converts the array to an `mlreportgen.dom.UnorderedList` object, appends the object to the document, and returns the object, which you can then format. In the cell array, you can include strings, numbers, and some DOM objects, such as a `Text` object. For a list of DOM objects you can include, see mlreportgen.dom.ListItem.

```
import mlreportgen.dom.*;
d = Document('myListReport','html');

t = Text('third item');
append(d,{'first item',6,t,'fourth item'});

close(d);
rptview(d.OutputPath);
```

Create a List Using an Array

You can create an unordered or ordered list from an array by including the array in an `UnorderedList` or `OrderedList` object constructor. In the cell array, you can include strings, numbers, and some DOM objects, such as a `Text` object. For a list of DOM objects you can include, see mlreportgen.dom.ListItem.

This example creates an unordered list. Change the `UnorderedList` class to `OrderedList` to number the items.

```
import mlreportgen.dom.*;
d = Document('unorderedListReport','html');

ul = UnorderedList({Text('item1'),'item 2',3});
append(d,ul);

close(d);
rptview(d.OutputPath);
```

Create a List Item by Item

You can create a list one item at a time by creating `mlreportgen.dom.ListItem` objects and appending them to an `UnorderedList` or `OrderedList` object.

This example creates an ordered list. Change the `OrderedList` class to `UnorderedList` to use bullet items.

```
import mlreportgen.dom.*;
d = Document('unorderedListReport','html');

li1 = ListItem('Rank 3 magic square:');
table = append(li1,Table(magic(3)));
table.Border = 'inset';
table.Width = '1in';
li2 = ListItem('Second item');
li3 = ListItem('Third item');

ul = OrderedList();
append(ul,li1);
append(ul,li2);
append(ul,li3);

append(d,ul);
```

```
close(d);
rptview(d.OutputPath);
```

Create a Multilevel List

A multilevel list is an ordered or unordered list whose list items contain ordered or unordered lists. You can create lists that have as many as nine levels.

You can create multilevel lists either from cell arrays or one list at a time. Creating a multilevel list one item at a time is useful for creating list items that contain multiple paragraphs, paragraphs and tables, and other combinations of document elements.

Create a Multilevel List from a Cell Array

You can use any of these approaches to create a multilevel list from a cell array.

- Nest one-dimensional cell arrays representing sublists in a one-dimension cell array representing the parent list.

  ```
  import mlreportgen.dom.*;
  d = Document('orderedListReport','html');

  ol = OrderedList({'step 1','step 2',...
       {'option 1','option 2'},...
       'step 3'});
  append(d,ol);

  close(d);
  rptview(d.OutputPath);
  ```

- Include list objects as members of a one-dimensional cell array representing the parent list. Use this approach to create ordered sublists from cell arrays.

  ```
  d = Document('myListReport','html');

  append(d,{'1st item',OrderedList({'step 1','step 2'}),'2nd item'});

  close(d);
  rptview(d.OutputPath);
  ```

- Combine the nested cell array and nested list object approaches.

Create a Multilevel List One List at a Time

You can create a multilevel list from scratch by appending child lists to parent lists.

```
import mlreportgen.dom.*;
d = Document('orderedListReport','html');

ol = OrderedList({'Start MATLAB', ...
     'Create a rank 3 or 4 magic square:'});
optionList = UnorderedList;
li = ListItem('>> magic(3)');
table = append(li,Table(magic(3)));
table.Width = '1in';
append(optionList, li);
li = ListItem('>> magic(4)');
table = append(li,Table(magic(4)));
table.Width = '1in';
append(optionList,li);
append(ol, optionList);
append(ol, ListItem('Close MATLAB'));,
append(d,ol);
close(d);
rptview('orderedListReport','html');
```

Format Lists

You can use list styles defined in a report style sheet to specify the indentation of each level of a list and the type of bullet or the number format used to render list items. For PDF and HTML, you can specify the bullet type or numbering type in the style sheet or using the mlreportgen.dom.ListStyleType format property on a `ListItem` object.

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:

```
import mlreportgen.dom.*;
d = Document('myListReport','html','MyTemplate');

list = append(d,{'first item',...
    OrderedList({'step 1','step 2'}),'second item'});
list.StyleName = 'MyListStyle';

close(d);
rptview('myListReport','html');
```

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 `UnorderedList` object that numbers top-level items, the top-level items are numbered, even though the object type is unordered (bulleted).

Create a Word List Style

To define a list style in a Word template, select `List` as the style type in the Create New Style from Formatting dialog box. See "Add Styles to a Word Template" on page 13-131.

Create an HTML or PDF List Style

To define a list style in an HTML or PDF template cascading style sheet (CSS), 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.

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.

See Also

Classes

mlreportgen.dom.ListItem | mlreportgen.dom.ListStyleType | mlreportgen.dom.OrderedList | mlreportgen.dom.UnorderedList

Functions
mlreportgen.dom.OrderedList.append

Related Examples

- “Use Style Sheet Styles” on page 13-24

Create and Format Tables

In this section...
“Two Types of Tables” on page 13-69
“Create a Table from a Two-Dimensional Array” on page 13-70
“Create a Table Using the Table entry Function” on page 13-70
“Create a Table from Scratch” on page 13-71
“Format a Table” on page 13-72
“Create a Formal Table” on page 13-77
“Format a Formal Table” on page 13-77
“Create and Format Table Rows” on page 13-78
“Format Table Columns” on page 13-79
“Create and Format Table Entries” on page 13-80

Two Types of Tables

You can use the DOM API to create two types of tables that differ in structure.

- An informal table (i.e., a table) consists of rows that contain table entries.
- A formal table contains a header, a body, and a footer section. Each section contains rows that contain table entries.

Informal tables are useful for most of your reporting needs. Use formal tables for tables whose headers or footers contain multiple rows.

For details about informal tables, see:

- “Create a Table from a Two-Dimensional Array” on page 13-70
- “Create a Table Using the Table entry Function” on page 13-70
- “Create a Table from Scratch” on page 13-71
- “Format a Table” on page 13-72

For details about formal tables, see:

- “Create a Formal Table” on page 13-77

- “Format a Formal Table” on page 13-77

Create a Table from a Two-Dimensional Array

You can create a table by appending a two-dimensional numeric array or a cell array containing built-in MATLAB data (strings and numbers) and DOM objects (`Text`, `Table`, `Image`, etc.) to a document. The `append` function converts the array to a `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 including a two-dimensional array in its constructor.

This example shows how to create a table from a numeric array and another table from a cell array of various object types. The cell array contains a magic square, which is rendered as an inner table. The cell array also includes a `Text` object constructor that uses the `AlertLevel` template style.

```
import mlreportgen.dom.*;
doc = Document('test');

table1 = append(doc,magic(5));
table1.Border = 'single';
table1.ColSep = 'single';
table1.RowSep = 'single';

ca = {'text entry',Paragraph('a paragraph entry'); ...
       Text('Danger!','AlertLevel'),magic(4)};
table2 = Table(ca);
append(doc,table2);

close(doc);
rptview(doc.OutputPath);
```

Create a Table Using the Table entry Function

You can use the `entry` function with a `Table` object to add content to a table entry or to format an entry. This approach is useful when you need to format table entries individually. For example:

```
import mlreportgen.dom.*;
doc = Document('test');

a = magic(5);
```

```
[v,i] = max(a);
[v1,i1] = max(max(a));
table = Table(a);

text = table.entry(i(i1),i1).Children(1);
text.Color = 'red';
append(doc,table);

close(doc);
rptview(doc.OutputPath);
```

Create a Table from Scratch

You can create a table from scratch by creating `TableEntry` objects, appending them to `TableRow` objects, and appending the `TableRow` objects to a `Table` object. This approach is useful when you need to create table entries that span multiple columns or rows that have a different number of entries. This example shows how to create a table with four columns and two rows. In the first table row, the second entry spans the second and third columns.

```
import mlreportgen.dom.*;
doc = Document('test');

table = Table(4);
table.Border = 'single';
table.ColSep = 'single';
table.RowSep = 'single';

row = TableRow;
append(row, TableEntry('entry 11'));
te = TableEntry('entry 12-13');
te.ColSpan = 2;
te.Border = 'single';
append(row, te);
append(row, TableEntry('entry 14'));
append(table,row);

row = TableRow;
for c = 1:4
  append(row, TableEntry(sprintf('entry 2%i', c)));
end
append(table,row);
```

```
append(doc,table);

close(doc);
rptview(doc.OutputPath);
```

Format a Table

You can format a table programmatically, 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 13-21.

Format a Table Programmatically

You can use format objects to format tables or use `Table` format properties to specify commonly used table formats. This example uses:

- `Border`, `ColSep`, and `RowSep` format objects to specify a red table border and the green column and row separators
- The `Width` format property to specify the table width

```
import mlreportgen.dom.*;
d = Document('test','html');

table = Table(magic(5));
table.Style = {Border('inset','red','3px'), ...
               ColSep('single','green','1px'), ...
               RowSep('single','green','1px')};

table.Width = '50%';

append(d, table);

close(d);
rptview(d.OutputPath);
```

Use these format objects and format properties to format a table.

Formatting	Format Object	Format Property
Width of table	`Width`	`Width`
Color of table background	`BackgroundColor`	`BackgroundColor`
Create border around table	`Border`	`Border`

Formatting	Format Object	Format Property
Color of border	`Border`	`BorderColor`
Thickness of border	`Border`	`BorderWidth`
Create left, right, top, or bottom table border	`Border`	n/a
Collapse table and table entry borders (HTML)	`BorderCollapse`	`BorderCollapse`
Create column separator	`ColSep`	`ColSep`
Column separator color	`ColSep`	`ColSepColor`
Column separator thickness	`ColSep`	`ColSepWidth`
Create row separator	`RowSep`	`RowSep`
Row separator color	`RowSep`	`RowSepColor`
Row separator thickness	`RowSep`	`RowSepWidth`
Indent table from left margin	`OuterMargin`	`OuterLeftMargin`
Space before or after table	`OuterMargin`	n/a
Space to right of table	`OuterMargin`	n/a
Align table left, right, or center	`HAlign`	`HAlign`
Specify table entry flow direction (left-to-right or right-to-left)	`FlowDirection`	`FlowDirection`
Resize table columns to fit contents	`ResizeToFitContents`	n/a

Format Table Entries

A `Table` object has properties that allow you to specify the same format or set of formats for all its entries.

Formatting	Table Object Property
Align entries vertically (top, middle, bottom)	`TableEntriesValign`

Formatting	Table Object Property
Align entries horizontally (left, right, center)	`TableEntriesHalign`
Create space (padding) between entry boundary and content	`TableEntriesInnerMargin`
Apply a set of format objects to all table entries	`TableEntriesStyle`

You can use a mlreportgen.dom.TextOrientation format object to make the text in a table entry vertical or horizontal.

Keep a Table and Title on the Same Page

Use the `KeepLinesTogether` and `KeepWithNext` paragraph formats to keep a table title and the table together on the same page. This example creates a table title, creates table content, and makes the table header row bold, using table entry indexing. To keep the table on the same page, the code specifies `KeepLinesTogether` and `KeepWithNext` for all rows except the last row. The last row has only `KeepLinesTogether` set and not `KeepWithNext`. This prevents the table from being forced to stay with the paragraph that follows.

```
import mlreportgen.dom.*
rpt = Document('test','docx');

p = Paragraph('Table 1');
p.Style = {Bold,KeepLinesTogether,KeepWithNext};
append(rpt, p);

ca = {Paragraph('Col 1'),Paragraph('Col 2'); ...
      Paragraph('data 11'),Paragraph('Data 12'); ...
      Paragraph('data 21'),Paragraph('Data 22')};

ca{1,1}.Children(1).Bold = true;
ca{1,2}.Children(1).Bold = true;

for r = 1:2
    for c = 1:2
        ca{r, c}.Style = {KeepLinesTogether,KeepWithNext};
    end
end
```

```
for c = 1:2
    ca{3, c}.Style = {KeepLinesTogether};
end

 append(rpt, ca);

close(rpt);
rptview(rpt.OutputPath);
```

Format a Table Using Microsoft Word Style Sheets

You can format tables using an existing Word style in a template or a template style that you modify or add.

To define a table 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 `Table`.

For more information about using Word styles with DOM objects, see “Modify Styles in a Microsoft Word Template” on page 13-130.

Format an HTML or PDF Table Using a Style Sheet

You can format HTML and PDF tables using a CSS style defined in a template.

To define a table style in an HTML or PDF template, use a selector on a `table` style. For example:

```
table.MyTable {
  border-bottom-color: rgb(128, 128, 128);
  border-bottom-width: thin;
  border-collapse: collapse;
}
```

Tip Use the CSS child selector (`>`) to specify the format of the 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 > tr > td {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 11pt;
  text-align: center;
}
```

Apply a Style to a Table

Once you have defined a table style in a template, you can apply it to a `Table` object in your report program either as the second argument in the `Table` object constructor or by setting it to the `StyleName` property of the `Table` 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 or 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 any 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);
```

Create a Formal Table

To create a formal table, use the same basic approaches as with an informal table, except that you use an `mlreportgen.domlFormalTable` constructor to construct a formal table. The constructor optionally accepts a two-dimensional numeric array or a cell array of MATLAB data for the body, header, and footer sections.

If you build a formal table completely or partially from scratch, you can use the `FormalTable` object functions `appendHeaderRow` and `appendBodyRow` to append rows to the table header and footer sections. The `FormalTable.append` function appends a row to the body section. Alternatively, you can access a section using the `Header`, `Body`, or `Footer` properties of the `FormalTable` object.

```
import mlreportgen.dom.*
d = Document('test');

t = FormalTable({'a','b';'c','d'});

r = TableRow();
append(r,TableEntry('Column 1'));
append(r,TableEntry('Column 2'));
append(t.Header,r);

append(d,t);

close(d);
rptview(d.OutputPath);
```

Format a Formal Table

You can format a formal table programmatically, 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 13-21.

Format a Formal Table Programmatically

You can format a formal table programmatically the same way you format an informal table. The format objects and properties that apply to an informal table also apply to formal tables. In addition, you can format the header, body, and footer sections of an informal table programmatically. If you specify a format for the table and one of its

sections, the value you specify for the section overrides the value you specify for the table as a whole. Not all formal table formats apply to formal table sections. For example, you cannot indent a header, body, or footer section independently of the containing table. In other words, the `OuterLeftMargin` property does not apply to formal table sections.

Apply Table Styles to a Formal Table and Its Sections

Use the same procedure for defining formal table styles in templates as you use for defining informal table styles.

You can apply a table style to a formal table and to each of its sections. If you apply a table style to the table itself and to one of its sections (for example, the header), the section style overrides the table style.

Note: 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.

Create and Format Table Rows

If you want to build a table from scratch, you can use the `TableRow` constructor to create the rows. Format the rows and then append the rows to a table that you are building.

Create a Table Row

The `mlreportgen.dom.TableRow` constructor takes no arguments and returns a `TableRow` object. You can then create and append `TableEntry` objects to the object to complete the row construction. Once you construct the row, you can add the row to the table, using the `append` function. This example creates a two-column table with two rows.

```
import mlreportgen.dom.*
rpt = Document('test');

table = Table(2);

row = TableRow();
append(row,TableEntry('Col1'));
append(row,TableEntry('Col2'));
append(table,row);
```

```
row = TableRow();
append(row,TableEntry('data11'));
append(row,TableEntry('data12'));
append(table,row);

append(rpt,table);

close(rpt);
rptview(rpt.OutputPath);
```

Format a Table Row

Use these format objects and format properties to format a table row.

Row Height Formatting	Format Object	Format Property
Specify the exact height of a row	`RowHeight`	`Height`
Specify the minimum height of row (Word only)	`RowHeight`	n/a
Cause this row to repeat as header row when a table flows across pages	`RepeatAsHeaderRow`	n/a
Allow this row to straddle a page boundary	`AllowBreakAcrossPages`	n/a

Format Table Columns

To format table columns, you can use `mlreportgen.dom.TableColSpecGroup` objects, either alone or with `mlreportgen.dom.TableColSpecGroup` objects. Use a `TableColSpecGroup` object to specify the format of a group of adjacent table columns. Use a `TableColSpec` object to override, for some table columns, some or all the formats of a column group. In this example, the `TableColSpecGroup` property specifies a column width of 0.2 inches and green text. The `TableColSpec` overrides those formats for the first column, specifying a width of 0.5 inches and bold, red text.

```
import mlreportgen.dom.*
rpt = Document('test');

rank = 5;
```

```
table = Table(magic(rank));
table.Border = 'single';
table.BorderWidth = '1px';

grps(1) = TableColSpecGroup;
grps(1).Span = rank;
grps(1).Style = {Width('0.2in'),Color('green')};

specs(1) = TableColSpec;
specs(1).Span = 1;
specs(1).Style = {Width('0.5in'),Bold,Color('red')};

grps(1).ColSpecs = specs;

table.ColSpecGroups = grps;
append(rpt,table);

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.

Create and Format Table Entries

If you need to build a table from scratch, you can use the `mlreportgen.dom.TableEntry` constructor to create table entries. You can then format the table entries and add them to table rows, which you can then add to the table you are building. If you need to format entries in a table that you have created from a cell array, you can use the `TableEntry` or `TableRow` function `entry` to gain access to an entry, which you can then format.

Create a Table Entry

Use a `TableEntry` constructor to create a table entry. You can optionally use the constructor to specify these kinds of entry content:

- Char array
- Any of these kinds of DOM objects:
  - `Paragraph`
  - `Text`

- `Image`
- `Table`
- `OrderedList`
- `UnorderedList`
- `CustomElement`

Format Table Entries Programmatically

You can use format objects or `TableEntry` format properties to format a table entry programmatically.

Formatting	Format Object	Format Property
Create border around entry	`Border`	`Border`
Color of border	`Border`	`BorderColor`
Thickness of border	`Border`	`BorderWidth`
Create left, right, top, or bottom entry border	`Border`	n/a
Align entry content top, bottom, middle	`VAlign`	`VAlign`
Space between entry boundary and entry content	`InnerMargin`	`InnerMargin`
Space between entry content and its top, bottom, right, or left boundaries	`InnerMargin`	n/a
Cause entry to span multiple columns	`ColSpan`	`ColSpan`
Cause entry to span multiple rows	`RowSpan`	`RowSpan`

You can 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 px'));
```

Properties you set for a `TableEntry` object take precedence over `TableEntriesStyle` format objects.

Format Table Entries Using Style Sheets

For HTML 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');
```

See Also

Classes

mlreportgen.dom.AllowBreakAcrossPages | mlreportgen.dom.ColSep | mlreportgen.dom.FlowDirection | mlreportgen.dom.FormalTable | mlreportgen.dom.RepeatAsHeaderRow | mlreportgen.dom.ResizeToFitContents | mlreportgen.dom.RowHeight | mlreportgen.dom.RowSep | mlreportgen.dom.Table | mlreportgen.dom.TableBody | mlreportgen.dom.TableColSpec | mlreportgen.dom.TableColSpecGroup | mlreportgen.dom.TableEntry | mlreportgen.dom.TableFooter | mlreportgen.dom.TableHeader | mlreportgen.dom.TableHeaderEntry | mlreportgen.dom.TableRow

Functions

mlreportgen.dom.FormalTable.appendFooterRow | mlreportgen.dom.FormalTable.appendHeaderRow | mlreportgen.dom.TableRow.append

Related Examples

- “Add Content to a Report” on page 13-11

More About

- “Report Formatting Approaches” on page 13-21

Create Links

In this section...
“Create a Link Target” on page 13-83
“Create an External Link” on page 13-83
“Create an Internal Link” on page 13-84
“Add Text or Images to Links” on page 13-84
“Create a Page Reference” on page 13-84

You can add these kinds of links to a report:

- External — Link to a location outside of the report, such as an HTML page or a PDF file. Use an `mlreportgen.dom.ExternalLink` object.
- Internal — Link to locations in the report. Use an `mlreportgen.dom.InternalLink` object.

Create a Link Target

To specify the link target for an `InternalLink` object, use the value in the `Name` property of an `mlreportgen.dom.LinkTarget` object. When you construct an `ExternalLink` object, you can use a `LinkTarget` object `Name` value or a URL.

This example creates a link target called `home`, and uses `home` as the target for an internal link.

```
import mlreportgen.dom.*
d = Document('mydoc');

append(d,LinkTarget('home'));
append(d,InternalLink('home','Go to Top'));

close(d);
rptview(d.OutputPath);
```

Create an External Link

Use an `mlreportgen.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('http://www.mathworks.com/','MathWorks'));

close(d);
rptview('mydoc','html');
```

Create an Internal Link

To set up links to a location in a report, append an `mlreportgen.dom.InternalLink` object to the document or document element. Use an `mlreportgen.dom.LinkTarget` object with the document element to link to. For example, you can include an `About the Author` link to a section that has the heading `Author's Biography`.

```
import mlreportgen.dom.*
d = Document('mydoc');

append(d,InternalLink('bio','About the Author'));
h = Heading(1,LinkTarget('bio'));
append(h,'Author''s Biography');
append(d,h);

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.ExternalLink.append | mlreportgen.dom.InternalLink | mlreportgen.dom.LinkTarget | mlreportgen.dom.PageRef

Related Examples

- “Create Image Maps” on page 13-96
- “Add Content to a Report” on page 13-11

More About

- “Report Formatting Approaches” on page 13-21

Create and Format Images

In this section...
“Create an Image” on page 13-86
“Resize an Image” on page 13-87
“Image Storage” on page 13-87
“Links from an Image” on page 13-87

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 13-96.

See Also

mlreportgen.dom.Height | mlreportgen.dom.Image | mlreportgen.dom.ScaleToFit | mlreportgen.dom.Width

Related Examples

- “Add Content to a Report” on page 13-11

More About

- “Report Formatting Approaches” on page 13-21

Create a Table of Contents

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:

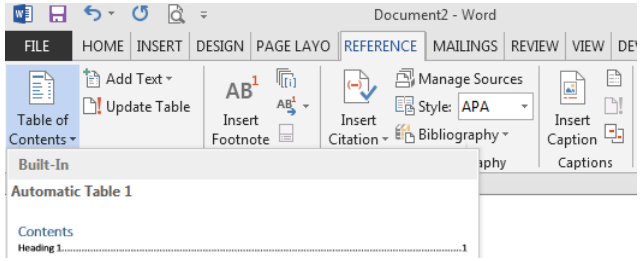
1. To specify where in the report to generate the TOC, create a table of contents reference in the Word template. See “Create a Word Table of Contents Reference” on page 13-89.
2. Set the outline levels of the section heads that you want to appear in the table of contents. See “Set Outline Levels of Section Heads” on page 13-93.
3. Update the generated document. See “Update the TOC in a Word Report” on page 13-91.

Create a Word Table of Contents Reference

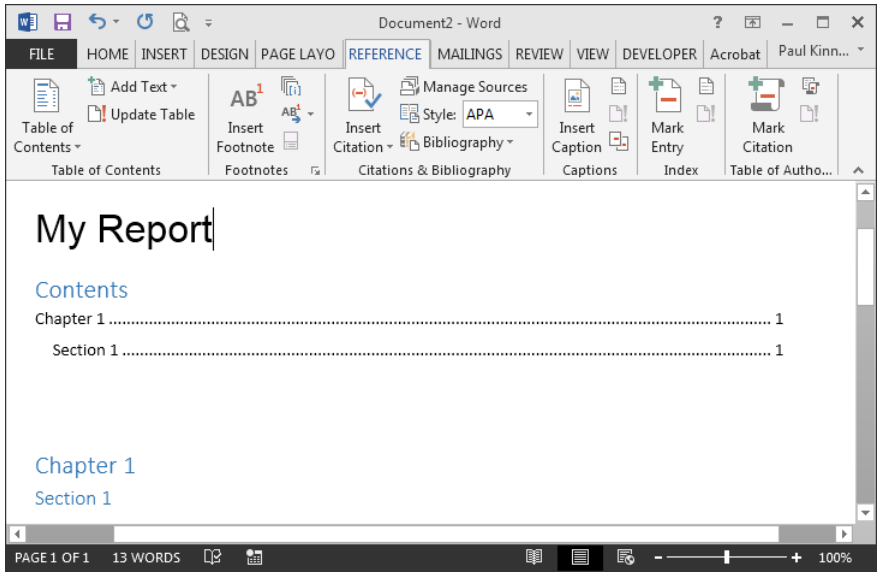
1. Open the template in Word.

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.



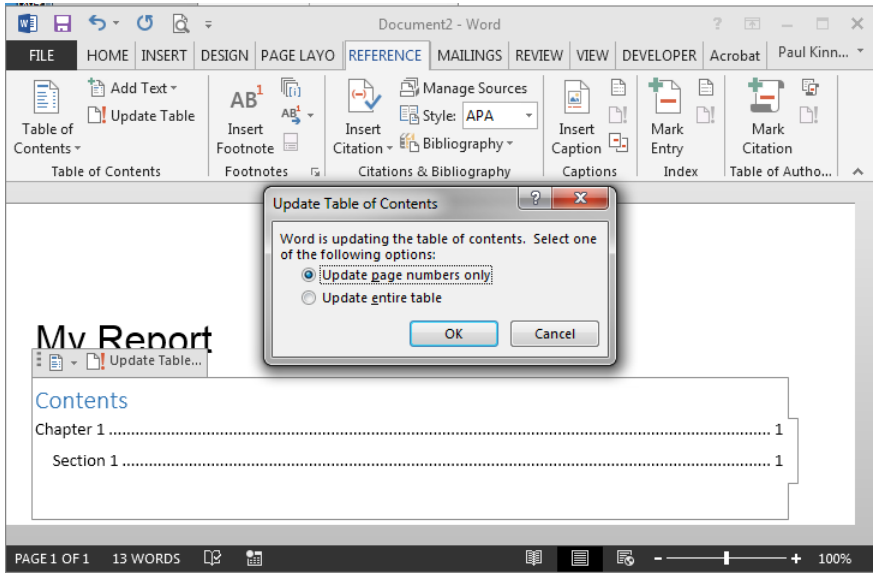
5 Save the template.

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 13-37 and “Insert TOC Placeholder Programmatically Using a Document Part” on page 13-92.

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 13-139.

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 uses 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 13-93
- "Use Format Objects to Set Outline Levels" on page 13-94
- "Use Heading Objects to Set Outline Levels" on page 13-94

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

`rptview` | `unzipTemplate` | `zipTemplate`

Classes

mlreportgen.dom.Heading | mlreportgen.dom.Heading1

Related Examples

- “Create a Microsoft Word Template” on page 13-124
- “Create an HTML or PDF Template” on page 13-135

Create Image Maps

In an HTML 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','html');

x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y);
annotation('textbox',[0.2,0.4,0.1,0.1],...
           'String','Help for plot function');
saveas(gcf,'plot_img.png');

plot1 = Image('plot_img.png');
append(d,plot1);

area1 = ImageArea(...
    ['http://www.mathworks.com/help/matlab/ref/' ...
    'plot.html?searchHighlight=plot'], ...
    'plot function help',240,450,463,492);

map = ImageMap();
plot1.Map = map;
append(map,area1);
```

```
close(d);
rptview(d.OutputPath);
```

See Also

Classes

mlreportgen.dom.Image | mlreportgen.dom.ImageArea | mlreportgen.dom.ImageMap

Functions

Related Examples

- “Add Content to a Report” on page 13-11

More About

- “Report Formatting Approaches” on page 13-21

Automatically Number Document Content

In this section...
“Automatically Number Content Programmatically” on page 13-98
“Automatically Number Content Using Part Templates” on page 13-100

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 `mlreportgen.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 `mlreportgen.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 mlreportgen.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.

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 13-100, 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

mlreportgen.dom.Document.createAutoNumberStream

Classes

mlreportgen.dom.AutoNumber | mlreportgen.dom.AutoNumberStream | mlreportgen.dom.CounterInc | mlreportgen.dom.CounterReset

Appending HTML to DOM Reports

You can append HTML markup or the entire contents of an HTML file to a programmatic report written with the DOM API. Append HTML to:

- Convert an existing HTML report to a Microsoft Word or PDF report.

  You can append HTML markup for a report to a DOM report, which you can then generate in Word or PDF format.
- Add content based on HTML markup.

  You can append the HTML content to a DOM report. You can use the HTML content as a building block in a DOM report that includes other report elements.

Workflow for Appending HTML

Appending HTML to a DOM report involves these tasks.

1. In a DOM report, append HTML content or an HTML file to a document or document part.

   For example, this DOM code creates a Word report that displays `Hello World`, based on the HTML content that you append to the report.

   ```
   import mlreportgen.com.*;
   d = Document('MyReport','docx');
   addHTML(d,'<p style="color:blue"><b>Hello World</b></p>');
   ```

   An alternative approach is to create an `mlreportgen.dom.HTML` or `mlreportgen.dom.HTMLFile` object and append it to a DOM report.
2. If you receive any MATLAB error messages, fix the source HTML markup and append the HTML again.

   The HTML content that you append must be XML parsable. For a summary of those requirements and for a list of supported HTML elements and HTML CSS formats, see “HTML Code Requirements for DOM Reports” on page 13-113.

See Also

mlreportgen.dom.Document.addHTML | mlreportgen.dom.Document.addHTMLFile | mlreportgen.dom.HTML | mlreportgen.dom.HTMLFile

Related Examples

- “Append HTML Content to DOM Reports” on page 13-105
- “Append HTML File Contents to DOM Reports” on page 13-107
- “Use an HTML Cleanup Program” on page 13-109

More About

- “HTML Code Requirements for DOM Reports” on page 13-113

Append HTML Content to DOM Reports

In this section...
“Use an addHTML Method” on page 13-105
“Append an HTML Object” on page 13-106
“Address Errors” on page 13-106

You can append strings of HTML content to a DOM document or document part using either of these approaches:

- Use the `addHTML` method with a `Document` or `DocumentPart` object.
- Create and append an `HTML` object.

If the HTML content that you append does not meet DOM requirements, the DOM API generates error messages. You can use an HTML cleanup program such as HTML Tidy on a file containing the source HTML content. HTML Tidy fixes many issues and also identifies issues you need to address manually. After you clean up the source HTML content, append it to a DOM report.

Use an addHTML Method

You can use the `addHTML` method with an `mlreportgen.dom.Document` or `mlreportgen.dom.DocumentPart` object to add a string of HTML content to a DOM report.

For example, you can use `addHTML` to create an `HTML` object that you append to a DOM report that produces Word output.

```
import mlreportgen.dom.*;
rpt = Document('HTMLToWordReport','docx');
htmlObj = addHTML(rpt,...
    '<p><b>Hello</b> <i style="color:green">World</i></p>');

close(rpt);
rptview(rpt.OutputPath);
```

Append an HTML Object

You can create an `mlreportgen.dom.HTML` object and append it to a DOM report. 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 cloned copy to the report.

For example, you can create an `HTML` object from HTML markup to use for a Word report.

```
import mlreportgen.dom.*;
rpt = Document('MyRep1','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);
```

Address Errors

If you receive any MATLAB error messages, fix the source HTML markup and append the HTML again. You can use an HTML cleanup program such as HTML Tidy on a file containing the source HTML content. HTML Tidy fixes many issues and also identifies issues you need to address manually. After you clean up the source HTML content, append it to a DOM report. For details, see “Use an HTML Cleanup Program” on page 13-109.

See Also

mlreportgen.dom.Document.addHTML | mlreportgen.dom.HTML

Related Examples

- “Append HTML File Contents to DOM Reports” on page 13-107
- “Use an HTML Cleanup Program” on page 13-109

More About

- “Appending HTML to DOM Reports” on page 13-103
- “HTML Code Requirements for DOM Reports” on page 13-113

Append HTML File Contents to DOM Reports

In this section...
“Use an addHTMLFile Method” on page 13-107
“Append an HTMLFile Object” on page 13-107
“Address Errors” on page 13-108

You can append HTML files to a DOM document or document part using either of these approaches:

- Use the `addHTMLFile` method with a `Document` or `DocumentPart` object.
- Create and append an `HTMLFile` object.

If the HTML file contents that you append do not meet DOM requirements, the DOM API generates error messages. You can use an HTML parser and cleanup program such as HTML Tidy to fix many issues and to identify issues you need to address manually.

Use an addHTMLFile Method

You can use the `addHTMLFile` method with an `mlreportgen.dom.Document` or `mlreportgen.dom.DocumentPart` object to add the contents of an HTML file to a DOM report.

For example, you can use `addHTMLFile` to create an `HTMLFile` object that you append to a DOM report that produces Word output.

```
import mlreportgen.dom.*;
rpt = Document('HTMLToWordReport','docx');
htmlObj = addHTML(rpt,...
    '<p><b>Hello</b> <i style="color:green">World</i></p>');

close(rpt);
rptview(rpt.OutputPath);
```

Append an HTMLFile Object

You can create an `mlreportgen.dom.HTMLFile` object and append it to a DOM report.

For example, you can convert the contents of two HTML files to a DOM report in Word format. This example assumes that there are HTML files called `myHTMLfile1.html` and `myHTMLfile2.html` in the current MATLAB folder.

```
import mlreportgen.dom.*;
rpt = Document('MyHTMLReport','docx');

path = 'myHTMLfile1.html';
htmlFile1 = HTMLFile(path);

htmlFile2 = HTMLFile('myHTMLFile2.html');
append(htmlFile1,htmlFile2)
append(rpt,htmlFile1);

close(rpt);
rptview(rpt.OutputPath);
```

Address Errors

If you receive any MATLAB error messages, fix the source HTML markup and append the HTML again. You can use an HTML cleanup program such as HTML Tidy on the HTML file to fix many issues. HTML Tidy also identifies issues you need to address manually. After you clean up the HTML content, append it to a DOM report. For details, see “Use an HTML Cleanup Program” on page 13-109.

See Also

mlreportgen.dom.Document.addHTMLFile | mlreportgen.dom.HTMLFile

Related Examples

- “Append HTML Content to DOM Reports” on page 13-105
- “Use an HTML Cleanup Program” on page 13-109

More About

- “Appending HTML to DOM Reports” on page 13-103
- “HTML Code Requirements for DOM Reports” on page 13-113

Use an HTML Cleanup Program

You can use an HTML cleanup program such as HTML Tidy to fix many issues and to identify issues you need to address manually. For a description of requirements for HTML content that you can append, see “HTML Code Requirements for DOM Reports” on page 13-113 .

Use HTML Tidy to Fix HTML Code

You can use the HTML Tidy program to fix HTML content so that it meets the requirements for appending to a DOM report. This example uses a batch file to fix the HTML content.

1 Copy this HTML content into a text editor such as WordPad.

```
<html>
<head>
  <title>Hi there</title>
</head>
<body>
  <p>This is a page
  a simple page with a simple table
<style>
table, th, td {
    border: 1px solid black;
}
</style>
<table style="width:50%">
  <tr>
    <td><b>Name</B></td>
    <td><b>Age</b></td>
    <td><b>Occupation</b></td>
  </tr>
  <tr>
    <td>Joe Smith</td>
    <td>40</td>
    <td>Plumber</td>
  </tr><tr>
    <td>Sue Jones</td>
    <td>33</td>
    <td>Scientist</td>
  </tr>
<tr>
```

```
    <td>Carlos Martinez</td>
    <td>38</td>
    <td>Lawyer</td>
  </tr>

</table>
  </body>
</html>
```

This HTML content has elements that are not XML parsable, including:

- Lack of a closing tag:

  ```
  <p>This is a page
    a simple page with a simple table
  ```

- Inconsistent case for an element tag:

  ```
  <td><b>Name</B></td>
  ```

2. In the current MATLAB folder, save the file using the file name `simple_html_example.html`.
3. Display the file in an HTML browser. Although the HTML content contains elements that are not XML parsable, it displays properly in most HTML browsers, such as Internet Explorer.

This is a page a simple page with a simple table

Name	Age	Occupation
Joe Smith	40	Plumber
Sue Jones	33	Scientist
Carlos Martinez	38	Lawyer

4. In MATLAB, try appending the HTML file to a DOM report.

```
import mlreportgen.dom.*;
rpt = Document('html_report','docx');
htmlFile = HTMLFile('simple_html_example.html');
```

You receive this error.

```
Error using mlreportgen.dom.HTMLFile
```

```
Parsing HTML text:
 "simple_html_example.html"
 caused error:
 "HTML error: "expected end of tag 'b'""
```

5 Download the HTML Tidy program. For example, to download Tidy for Windows, go to http://www.paehl.com/open_source/?HTML_Tidy_for_Windows. Click the `EXE Version compiled 06 nov 2009` link.

Note: To download Tidy for other platforms, see http://tidy.sourceforge.net/#binaries.

---

6 In the `tidy.zip` file, right-click `tidy.exe` and select **Extract**. Extract `tidy.exe` to the current MATLAB folder.

7 Create a batch file to use with Tidy. In Notepad, enter the following code.

```
tidy --doctype omit --input-xml no --output-xml yes --write-back yes -f errs.txt %1
```

Save the batch file in the Windows path. Save the file as `tidyup.bat`. You can use this batch file with other HTML files that you want to append to a DOM report.

8 Make a backup copy of the `simple_html_example.html` file, which contains the HTML to append to the DOM report.

9 Run Tidy on `simple_html_example.html`. In a Windows command window, enter:

```
tidyup simple_html_example.html
```

10 In the folder where you ran `tidyup`, check the `errs.txt` file. That file summarizes the changes Tidy made, and lists as errors issues that Tidy could not fix. In this example, there are no errors, but if `errs.txt` did report errors, manually edit the HTML file to address those issues.

11 In MATLAB, append the `simple_html_example.html` file to a DOM report and display the report.

```
import mlreportgen.dom.*;
rpt = Document('html_report','docx');
htmlFile = HTMLFile('simple_html_example.html');
append(rpt,htmlFile);

close(rpt);
rptview(rpt.OutputPath);
```

Related Examples

- “Append HTML Content to DOM Reports” on page 13-105

- “Append HTML File Contents to DOM Reports” on page 13-107

More About

- “HTML Code Requirements for DOM Reports” on page 13-113
- “Appending HTML to DOM Reports” on page 13-103

HTML Code Requirements for DOM Reports

In this section...
"XML-Parsable HTML Code" on page 13-113
"Supported HTML Elements and Attributes" on page 13-113
"Supported HTML CSS Style Attributes for All Elements" on page 13-115
"Support for HTML Character Entities" on page 13-117
"DOCTYPE Declaration" on page 13-117

XML-Parsable HTML Code

The HTML content that you append to a DOM report must be XML parsable. HTML content that 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 that 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.

Tip You can use the HTML Tidy program to ensure that your HTML file is XML parsable. For details, see "Use an HTML Cleanup Program" on page 13-109.

Supported HTML Elements and Attributes

In HTML content that you append to a DOM report, you can use these HTML elements and attributes.

HTML Element	Attributes
`a`	`class`, `style`, `href`, `name`

HTML Element	Attributes
b	class, style
body	class, style
br	n/a
code	class, style
del	class, style
div	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, alt
li	class, style
ol	class, style
p	class, style, align
pre	class, style
s	class, style
span	class, style
strike	class, style
sub	class, style
sup	class, style
table	class, style, align, bgcolor, border, cellspacing, cellpadding, frame, rules, width
tbody	class, style, align, valign
tfoot	class, style, align, valign
thead	class, style, align, valign

HTML Element	Attributes
`td`	`class`, `style`, `bgcolor`, `height`, `width`, `colspan`, `rowspan`, `valign`, `nowrap`
`tr`	`class`, `style`, `bgcolor`, `valign`
`tt`	`class`, `style`
`u`	`class`, `style`
`ul`	`class`, `style`

For information about these elements, see the W3Schools tags documentation at www.w3schools.com/tags.

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 CSS (cascading style sheets) formats.

These CSS formats are supported:

- `background-color`
- `border`
- `border-bottom`
- `border-bottom-color`
- `border-bottom-style`
- `boder-bottom-width`
- `border-color`
- `border-left`
- `border-left-color`
- `border-left-style`
- `boder-left-width`
- `border-right`
- `border-right-color`
- `border-rigtht-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, see the W3Schools CSS documentation at www.w3schools.com/cssref.

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 bracket (<) in HTML content that you want to append, use one of these character entity references:

- The named character entity reference `<`
- The numeric character entity reference `&003c;`

For more information, see http://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 http://www.w3schools.com/tags/tag_doctype.asp). 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 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, a valid DTD 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. 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.

Related Examples

- “Use an HTML Cleanup Program” on page 13-109
- “Append HTML Content to DOM Reports” on page 13-105
- “Append HTML File Contents to DOM Reports” on page 13-107

More About

- “Appending HTML to DOM Reports” on page 13-103

Display Report Generation Messages

In this section...
"Report Generation Messages" on page 13-119
"Display DOM Default Messages" on page 13-119
"Create and Display a Progress Message" on page 13-121

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;
   dispatcher.Filter.DebugMessagesPass = true;
   ```

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

mlreportgen.dom.MessageDispatcher.dispatch | mlreportgen.dom.MessageDispatcher.getTheDispatcher | mlreportgen.dom.ProgressMessage.formatAsHTML | mlreportgen.dom.ProgressMessage.formatAsText | mlreportgen.dom.ProgressMessage.passesFilter

Classes

mlreportgen.dom.DebugMessage | mlreportgen.dom.ErrorMessage | mlreportgen.dom.MessageDispatcher | mlreportgen.dom.MessageEventData | mlreportgen.dom.MessageFilter | mlreportgen.dom.ProgressMessage | mlreportgen.dom.WarningMessage

Compile a Report Program

If the MATLAB Compiler™ product is installed on your system, you can use it to compile your DOM-based report generation program. Compiling allows you to share your report generation program with others who do not have MATLAB installed on their systems.

To enable someone who does not have MATLAB installed to run your compiled program, your program must execute this statement before executing the first line of DOM code that it executes to generate a report:

```
makeDOMCompilable();
```

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.

See the Word documentation for information about how to create templates and to copy styles from one template to another.

Related Examples

- “Add Holes in a Microsoft Word Template” on page 13-125
- “Modify Styles in a Microsoft Word Template” on page 13-130
- “Create an HTML or PDF Template” on page 13-135

Add Holes in a Microsoft Word Template

In this section...
"Display the Developer Ribbon in Word" on page 13-125
"Inline and Block Holes" on page 13-125
"Create an Inline Hole" on page 13-126
"Create a Block-Level Hole" on page 13-126
"Set Default Text Style for a Hole" on page 13-127

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.

Display the Developer Ribbon in Word

To work with holes in a Word template, use the Word **Developer** ribbon. If the **Developer** ribbon does not appear, add it.

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`.

Inline and Block Holes

The DOM API supports two types of holes: inline and block.

- An inline hole is for document elements that a paragraph element can contain: `Text`, `Image`, `LinkTarget`, `ExternalLink`, `InternalLink`, `CharEntity`, and `AutoNumber`.

- A block hole can contain the same kinds of document elements as an inline hole, plus `Paragraph`, `Table`, `OrderedList`, `UnorderedList`, `DocumentPart`, and `Group` document elements.

Create an Inline Hole

1. Open the template in Word.
2. On the **Developer** ribbon, click **Design Mode**. This mode enables you to see the hole marks with the title tag after you create the hole.
3. Click in the paragraph where you want to create an inline hole.

Tip If the hole is the only hole in a paragraph or is at the end of a paragraph:

a. Add several blank spaces at the end of the paragraph.
b. Insert the hole before the spaces.
c. Delete the extra spaces.

---

4. Click the **Rich Text Control** button Aa. A rich text control appears at the insertion point.
5. Click the **Properties** button.
6. In the dialog box, in the **Title** field, enter an ID for the hole. In **Tag** field, enter `Hole`. Click **OK**. The hole ID appears on the rich text control.

Create a Block-Level Hole

The difference between creating a block-level hole and an inline hole is that the rich text content control must contain an empty paragraph rather than residing in a paragraph.

1. Open the Word template.
2. On the **Developer** ribbon, click **Design Mode**. This mode enables you to see the hole marks with the title tag after you create the hole.
3. Create an empty paragraph where you want to create a block-level hole. If you are at the end of a document, create a second empty paragraph.
4. Select the empty paragraph.
5. Click the **Rich Text Control** button Aa. A rich text control appears at the insertion point.

6 Click the **Properties** button.

7 In the dialog box, in the **Title** field, enter an ID for the hole. In the **Tag** field, enter `Hole`. Click **OK**. The hole ID appears on the rich text control.

Set Default Text Style for a Hole

Your template can specify the name of a default style to use to format `Text` and `Paragraph` objects appended to a hole. If such an object does not specify a style name, the DOM API sets the `StyleName` property to the default style, which must be a character or linked character and paragraph style defined in the template. Defining a default hole style eliminates the need to format hole content programmatically.

1 Open the Word template.

2 Select the **Developer** ribbon.

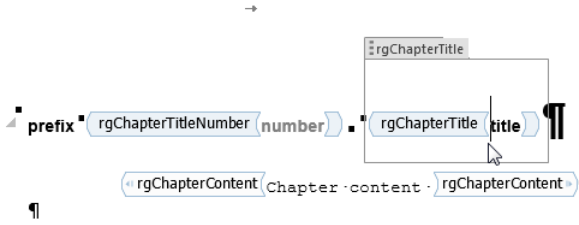
3 Click the hole whose default style name you want to specify.

If you have not created a hole, see "Inline and Block Holes" on page 13-125.

4 On the **Insert** ribbon, click 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.



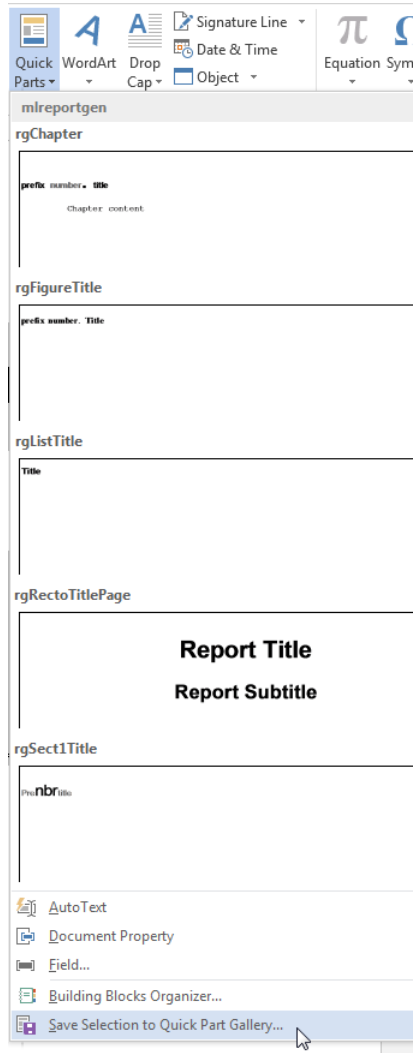
7 Select **Properties**.

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, or select **New Style** to create a style to use as the default style. 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 **Category** to `mlreportgen`. Click **OK**.

Related Examples

- “Modify Styles in a Microsoft Word Template” on page 13-130
- “Create an HTML or PDF Template” on page 13-135
- “Create and Format Tables” on page 13-69

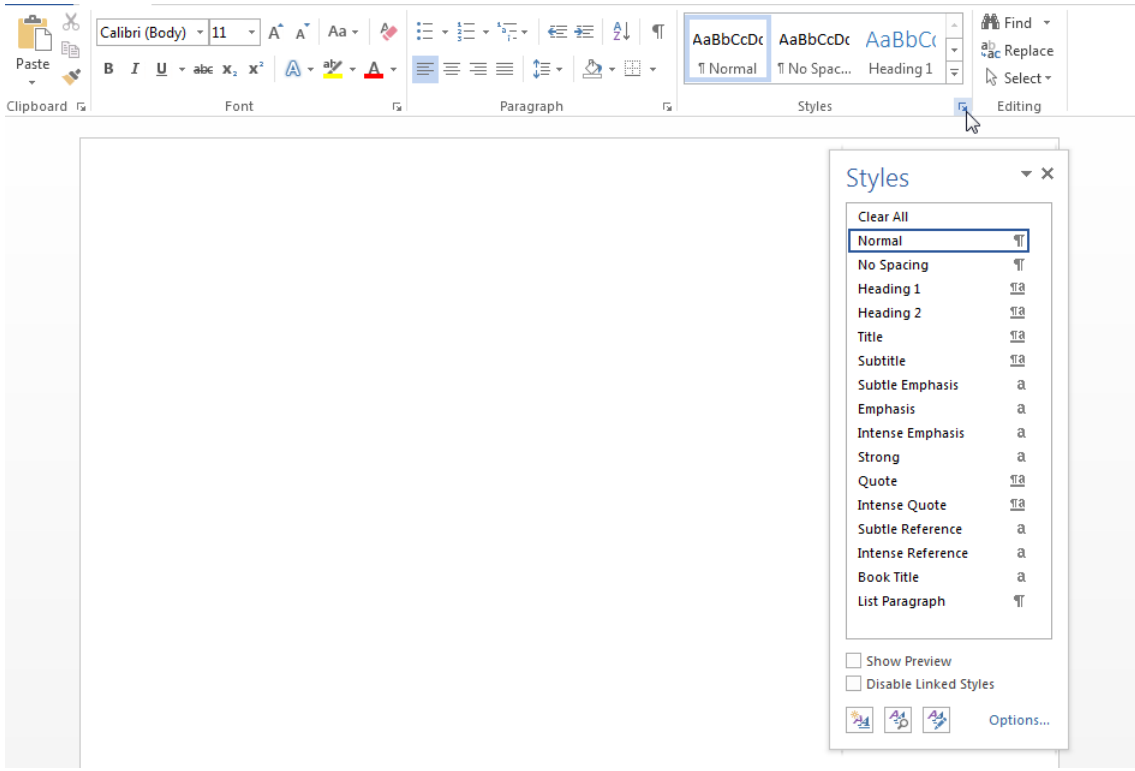
Modify Styles in a Microsoft Word Template

In this section...
“Edit Styles in a Word Template” on page 13-130 “Add Styles to a Word Template” on page 13-131

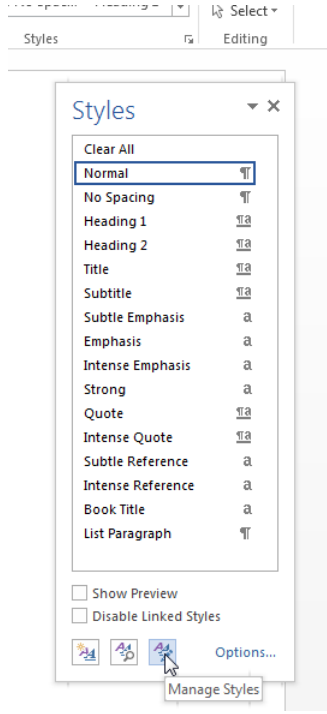
Edit Styles in a Word Template

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**.

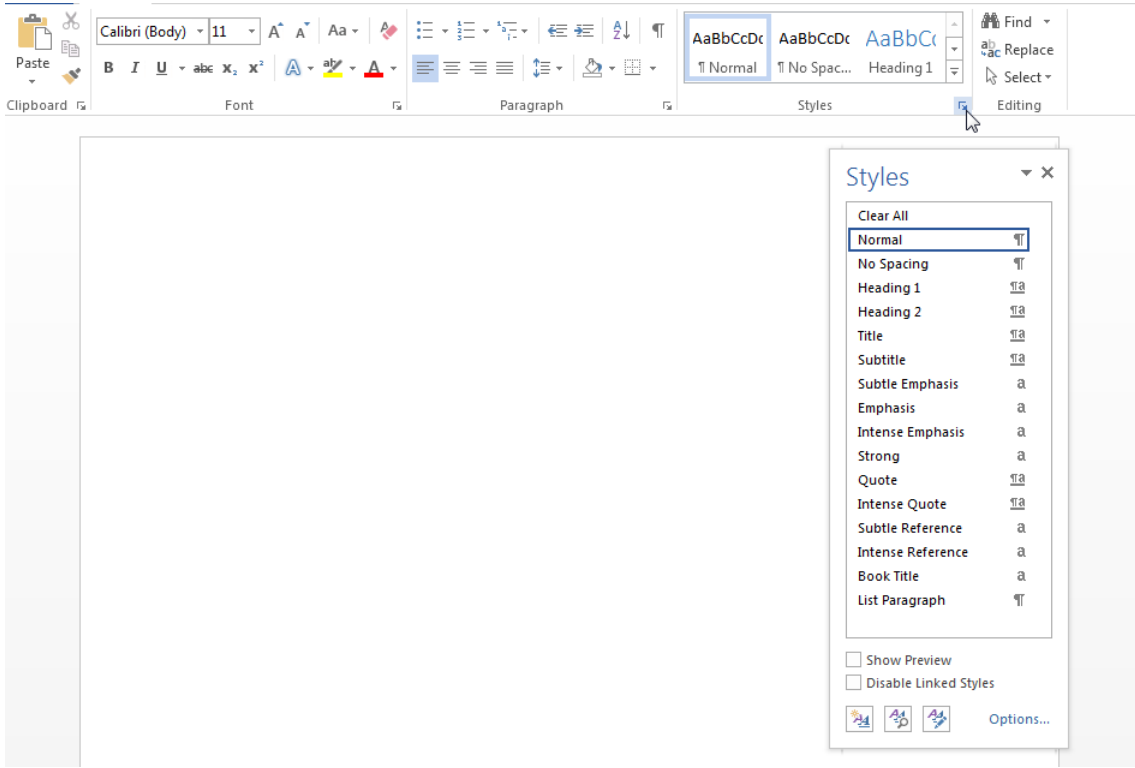
4 In the Modify Styles dialog box, change any of the style definitions. For example, change the font family, font size, or indentation. When you have finished with your changes, click **OK**, and then close the Manage Styles dialog box.

5 Save and close the template.

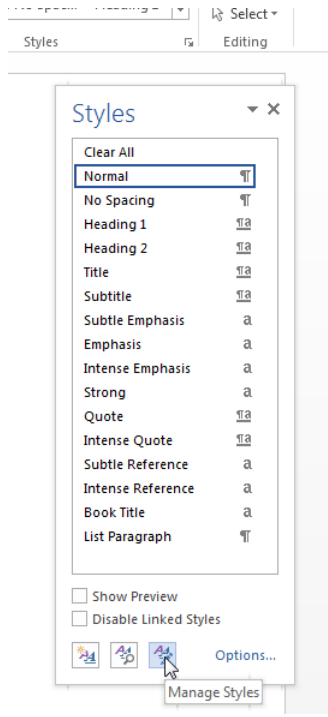
For more information about using Word styles, see the Microsoft Word documentation.

Add Styles to a Word Template

1 In your Word template, open the Styles pane.

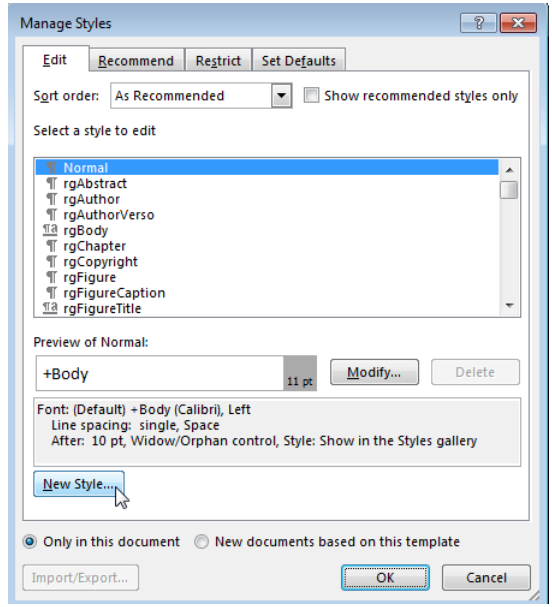


2 In the Styles pane, click the **Manage Styles** button.



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.

Related Examples

- "Add Holes in a Microsoft Word Template" on page 13-125
- "Create an HTML or PDF Template" on page 13-135

Create an HTML or PDF Template

Use one of these approaches to create an HTML or PDF template for generating a report.

- Use `mlreportgen.dom.Document.createTemplate` to create a copy of the DOM API default template that you can then customize. For example:

  ```
  mlreportgen.dom.Document.createTemplate('mytemplate','html');
  ```

- Create a template from scratch.

Edit an HTML or PDF Template

HTML and PDF templates are packaged in a zipped template package with the extension `.htmtx` for HTML and `.pdftx` for PDF. To edit one of these templates, unzip it to a folder using the `unzipTemplate` function. For example, to unzip a template called `mytemplate` in the current folder:

```
unzipTemplate('mytemplate')
```

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.htmtx')
```

For PDF, use the `.pdftx` extension:

```
zipTemplate('mytemplate.pdftx')
```

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. Make sure that the template is on the MATLAB path.

See Also

Functions

`unzipTemplate` | `zipTemplate`

Classes

mlreportgen.dom.Document

Related Examples

- “Add Holes in HTML and PDF Templates” on page 13-137
- “Modify Styles in HTML Templates” on page 13-143
- “Modify Styles in PDF Templates” on page 13-144
- “Create a Microsoft Word Template” on page 13-124

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 a string to use to identify the hole. 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

`unzipTemplate` | `zipTemplate`

Related Examples

- “PDF and HTML Document Parts and Holes” on page 13-139
- “Simplify Filling in Forms” on page 13-49
- “Fill the Blanks in a Report Form” on page 13-33
- “Modify Styles in HTML Templates” on page 13-143
- “Modify Styles in PDF Templates” on page 13-144
- “Create a Microsoft Word Template” on page 13-124

PDF and HTML Document Parts and Holes

This example shows how to:

- Define a document part template that has holes.
- Insert the document part 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.

   ```
   mlreportgen.dom.Document.createTemplate('myPDFtemplate','pdf');
   ```

2. Unzip the template package.

   ```
   unzipTemplate('myPDFtemplate.pdftx');
   ```

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" /><
         <hole id="rgChapterTitleNumber" default-style-name="rgChapterTitleNumber" /><
         <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.pdftx`.

```
zipTemplate('myPDFtemplate2.pdftx','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.

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.

Related Examples

- "Add Holes in HTML and PDF Templates" on page 13-137
- "Create a Microsoft Word Template" on page 13-124
- "Modify Styles in PDF Templates" on page 13-144

Modify Styles in PDF Templates

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 13-144.

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. For information on output modes, see “Stream a Report” on page 13-16.

These CSS formats are supported:

- `background-color`
- `border`
- `border-bottom`
- `border-bottom-color`
- `border-bottom-style`
- `boder-bottom-width`
- `border-color`
- `border-left`
- `border-left-color`
- `border-left-style`
- `boder-left-width`
- `border-right`
- `border-right-color`
- `border-rigtht-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, see the W3Schools CSS documentation at www.w3schools.com/cssref.

For information about FO, see w3.org/2002/08/XSLFOsummary.html.

Related Examples

- “Add Holes in HTML and PDF Templates” on page 13-137
- “Create a Microsoft Word Template” on page 13-124

- “Modify Styles in HTML Templates” on page 13-143

External Websites

- w3.org/2002/08/XSLFOsummary.html
- www.w3schools.com/cssref

Create Page Layout 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 of 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.

`style`	`page-margin: top bottom left right header footer gutter; page-size: height width orientation`
`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`

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 13-152.

```
<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);
```

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 a `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.DOCXPageLayout | mlreportgen.dom.PageMargins | mlreportgen.dom.PageSize | mlreportgen.dom.PDFPageLayout

Related Examples

- “Create a Microsoft Word Template” on page 13-124

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 13-157.

For details about working with Word page headers and footers, see the Word documentation.

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 13-37.

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 13-157.

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 `PagerHeader` 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 13-152 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 13-162.

See Also

Functions

mlreportgen.dom.Document.createTemplate

Classes

mlreportgen.dom.Document | mlreportgen.dom.DocumentPart | mlreportgen.dom.DOCXPageFooter | mlreportgen.dom.DOCXPageHeader | mlreportgen.dom.DOCXPageLayout | mlreportgen.dom.NumPages | mlreportgen.dom.Page | mlreportgen.dom.PDFPageFooter | mlreportgen.dom.PDFPageHeader | mlreportgen.dom.PDFPageLayout | mlreportgen.dom.StyleRef

Related Examples

- “Create a Microsoft Word Template” on page 13-124

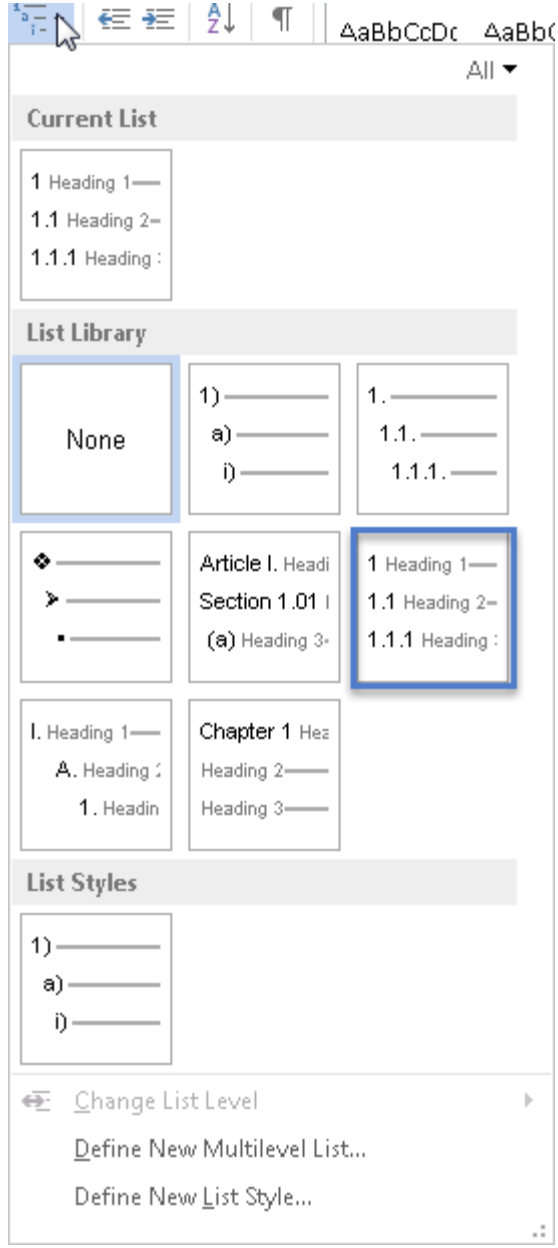
- “Create an HTML or PDF Template” on page 13-135
- “Add Complex Page Numbers in Microsoft Word” on page 13-162

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.


AaBbCcDc
AaBbC
All
Current List
1 Heading 1
1.1 Heading 2
1.1.1 Heading
List Library
None
1)
a)
i)
1.
1.1.
1.1.1.
Article I. Headi
Section 1.01
(a) Heading 3
1 Heading 1
1.1 Heading 2
1.1.1 Heading
I. Heading 1
A. Heading
1. Headin
Chapter 1 Hea
Heading 2
Heading 3
List Styles
1)
a)
i)
Change List Level
Define New Multilevel List...
Define New List Style...

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.DOCXPageFooter | mlreportgen.dom.DOCXPageLayout | mlreportgen.dom.PageNumber

14

Programmatic PowerPoint Presentation Creation

- "Create a Presentation Program" on page 14-2
- "Create PPT Objects" on page 14-7
- "Import the PPT API Package" on page 14-10
- "Get and Set PPT Object Properties" on page 14-11
- "Create a Presentation Object to Hold Content" on page 14-13
- "Generate a Presentation" on page 14-14
- "Display Presentation Generation Messages" on page 14-15
- "Compile a Presentation Program" on page 14-19
- "Presentation Formatting Approaches" on page 14-20
- "Presentation Format Inheritance" on page 14-24
- "Set Up a PowerPoint Template" on page 14-26
- "Access PowerPoint Template Elements" on page 14-37
- "Define a Style Using Format Objects" on page 14-43
- "Use Format Properties" on page 14-45
- "Update Presentation Content Programmatically" on page 14-48
- "Create a Presentation Programmatically" on page 14-59
- "Add Slides" on page 14-71
- "Add and Replace Presentation Content" on page 14-74
- "Create and Format Text" on page 14-81
- "Create and Format Paragraphs" on page 14-84
- "Create and Format Tables" on page 14-87
- "Create and Format Pictures" on page 14-96
- "Create and Format Links" on page 14-98

Create a Presentation Program

In this section...
“PPT API Programs” on page 14-3
“Two Ways to Use the PPT API” on page 14-4
“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.*;

slidesFile = 'mySlides.pptx';
slides = Presentation(slidesFile);

slide1 = add(slides,'Title Slide');
replace(slide1,'Title','My Presentation');
replace(slide1,'Subtitle','Create a Presentation Program');

slide2 = add(slides,'Title and Content');
para = Paragraph('First Content Slide');
para.FontColor = 'blue';
replace(slide2,'Title',para);

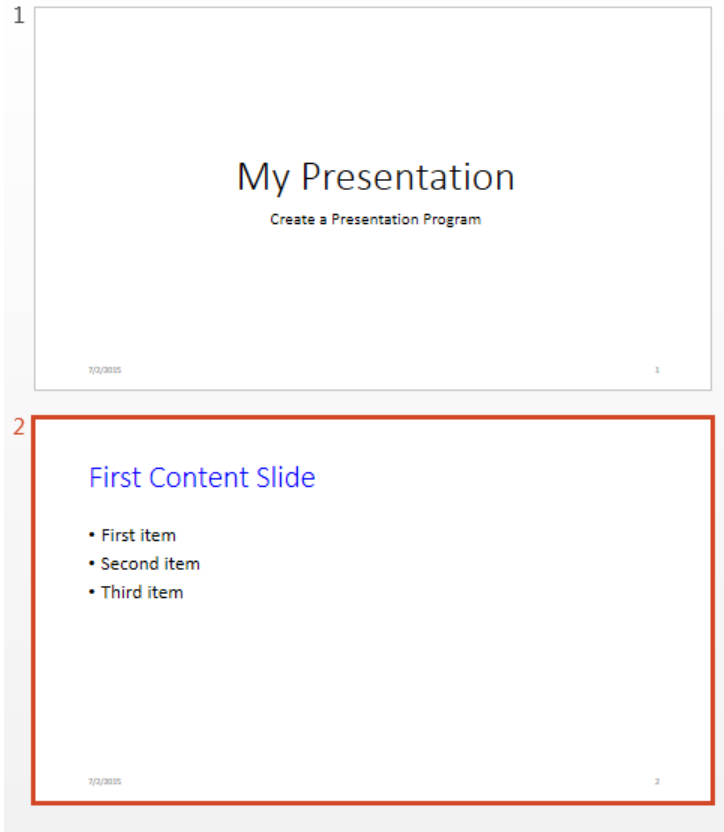
replace(slide2,'Content',{'First item','Second item','Third item'});

close(slides);
```

After you create the presentation, which is named `MySlides.pptx`, you can open it. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```

The generated presentation `MySlides.pptx` includes these two slides.



PPT API Programs

PPT API programs generally include code that:

- Imports the `mlreportgen.ppt` API package. Include an `import` statement. 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

```
slidesFile = 'mySlides.pptx';
slides = Presentation(slidesFile);
```

- Adds or replaces slide content.

```
contents = find(slide2,'Title');
replace(contents,Paragraph('First Content Slide'));

contents = find(slide2,'Content');
datePara = Paragraph('Today is ');
dateText = date;
append(datePara,dateText);
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(slides);
```

You can include code to open the presentation on Windows platforms. Use `winopen` with the name of the file, which in this case is stored in the `slidesFile` variable.

```
if ispc
    winopen(slidesFile);
end
```

To see another example of a PPT API program in MATLAB, enter `population_slides`.

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. 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
- An existing PowerPoint presentation whose content you want to update
- A PowerPoint template

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 PPT API comes with a default template that you can use to create presentations. If the default template does not meet your needs, you can use PowerPoint interactively to create templates that do meet your needs.

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.
Placeholders	Specifies an area of a slide layout that you can replace with a text string, 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.

Related Examples

- "Create PPT Objects" on page 14-7
- "Create a Presentation Object to Hold Content" on page 14-13
- "Update Presentation Content Programmatically" on page 14-48
- "Create a Presentation Programmatically" on page 14-59

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 14-8

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.*;
slides = Presentation('MySlides');

slide1 = add(slides,'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

mlreportgen.ppt.Presentation.add

Classes

mlreportgen.ppt.Presentation | mlreportgen.ppt.Slide

Related Examples

- "Import the PPT API Package" on page 14-10
- "Create a Presentation Object to Hold Content" on page 14-13

Import the PPT API Package

All PPT class names and constructor names have the prefix `mlreportgen.ppt`. To omit the prefix in your code, insert this statement at the beginning of a PPT API program.

```
import mlreportgen.ppt.*;
```

Examples that refer to PPT API objects and functions without the `mlreportgen.ppt` prefix assume that you have imported the PPT API package.

Related Examples

- “Create PPT Objects” on page 14-7
- “Get and Set PPT Object Properties” on page 14-11
- “Create a Presentation Program” 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'
```

Related Examples

- “Use Format Properties” on page 14-45
- “Create PPT Objects” on page 14-7

More About

- “Presentation Formatting Approaches” on page 14-20

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. To replace some of the content in an existing presentation but leave the rest, use the presentation as the template for the `Presentation` object you create.

When you create a complete presentation using the PPT API, use an empty presentation that has no slides or only a few slides.

See Also

mlreportgen.ppt.Presentation

Related Examples

- “Create a Presentation Program” on page 14-2
- “Create PPT Objects” on page 14-7
- “Generate a Presentation” on page 14-14

More About

- “Access PowerPoint Template Elements” on page 14-37

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.

Related Examples

- “Display Presentation Generation Messages” on page 14-15

Display Presentation Generation Messages

In this section...
“Presentation Generation Messages” on page 14-15
“Display PPT Default Messages” on page 14-15
“Create and Display a Progress Message” on page 14-17

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

mlreportgen.ppt.MessageDispatcher.dispatch | mlreportgen.ppt.MessageDispatcher.getTheDispatcher | mlreportgen.ppt.ProgressMessage.formatAsHTML | mlreportgen.ppt.ProgressMessage.formatAsText | mlreportgen.ppt.ProgressMessage.passesFilter

Classes

mlreportgen.ppt.DebugMessage | mlreportgen.ppt.ErrorMessage | mlreportgen.ppt.MessageDispatcher | mlreportgen.ppt.MessageEventData | mlreportgen.ppt.MessageFilter | mlreportgen.ppt.ProgressMessage | mlreportgen.ppt.WarningMessage

Compile a Presentation Program

If the MATLAB Compiler product is installed on your system, you can use it to compile your presentation program. Compiling allows you to share your report generation program with others who do not have MATLAB installed on their systems.

To enable someone who does not have MATLAB installed to run your compiled program, your program must execute this statement before the first line of PPT API code that the program executes to generate a report:

```
makePPTCompilable();
```

Presentation Formatting Approaches

In this section...
"Template Formatting" on page 14-21 "Format Objects" on page 14-21 "Format Properties" on page 14-22 "Interactive Formatting of Slide Content" on page 14-22

With the 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.	• Applying formatting globally within a presentation • Maintaining consistency across presentations • Extending formatting options that the PPT API provides
Using the PPT API, specify format objects to define a style for a presentation object.	• Formatting a specific presentation element • Specifying multiple format options in one statement • Specifying complicated values such as hexadecimal color values that are used repeatedly in a program • Extending formatting options beyond the ones that format properties of an object provide • Defining a style to use with multiple objects
Using the PPT API, set format properties of a presentation object.	• Specifying one or two basic format options for a specific presentation object • Extending formatting options beyond those options that format properties of an object provide • Specifying one or two basic format options for a specific presentation object

Formatting Approach	Use
In the PowerPoint software, format a generated PPT API.	• Customizing a specific version of a generated presentation • Extending formatting options beyond those options that the format objects provide

Template Formatting

Use templates for applying formatting globally:

- Across a whole presentation (for example, 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 presentations that use that 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:

```
p = Paragraph('Model Highlights');
p.Style = {FontColor('red'),Bold(true)};
```

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, using dot notation. For example:

```
p = Paragraph('My paragraph);
p.Bold = true;
```

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.

Related Examples

- “Set Up a PowerPoint Template” on page 14-26
- “Define a Style Using Format Objects” on page 14-43
- “Use Format Properties” on page 14-45

More About

- "Presentation Format Inheritance" on page 14-24
- "Access PowerPoint Template Elements" on page 14-37

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.*;

slidesFile = 'myParagraphPresentation.pptx';
slides = Presentation(slidesFile);
```

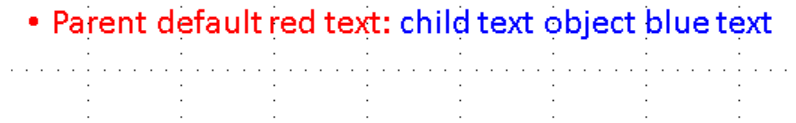
```
slide1 = add(slides,'Title and Content');

%% 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(slides);
```



Set Up a PowerPoint Template

In this section...
"Use Existing Presentations as Templates" on page 14-26
"Customize a Copy of the Default Template" on page 14-26
"Global Presentation Formatting Using a Slide Master" on page 14-27
"Add a Slide Master" on page 14-28
"Format a Slide Layout" on page 14-30
"Add a Slide Layout" on page 14-32
"Add a Placeholder" on page 14-33

Use Existing Presentations as Templates

When you use an existing PowerPoint presentation as a template for a PPT API presentation, the content from the template presentation appears in the new PPT API presentation. You can use the PPT API to update content in the existing presentation. You can also programmatically change some formatting of the content that you are updating.

To format a PPT API presentation that you create completely programmatically, specify an empty PowerPoint presentation as a template when you create a `Presentation` object.

Customize a Copy of the Default Template

You can use the default PPT API PowerPoint template as a starting point for a your own template.

Note: You can use a similar approach to customize a PowerPoint template other than the default PPT API template. To do so, when you create a `Presentation` object, specify the template that you want to customize.

---

1. In a PPT API program, create an empty `Presentation` object, without specifying a template. The PPT API uses the default PowerPoint template.
2. Generate the presentation.

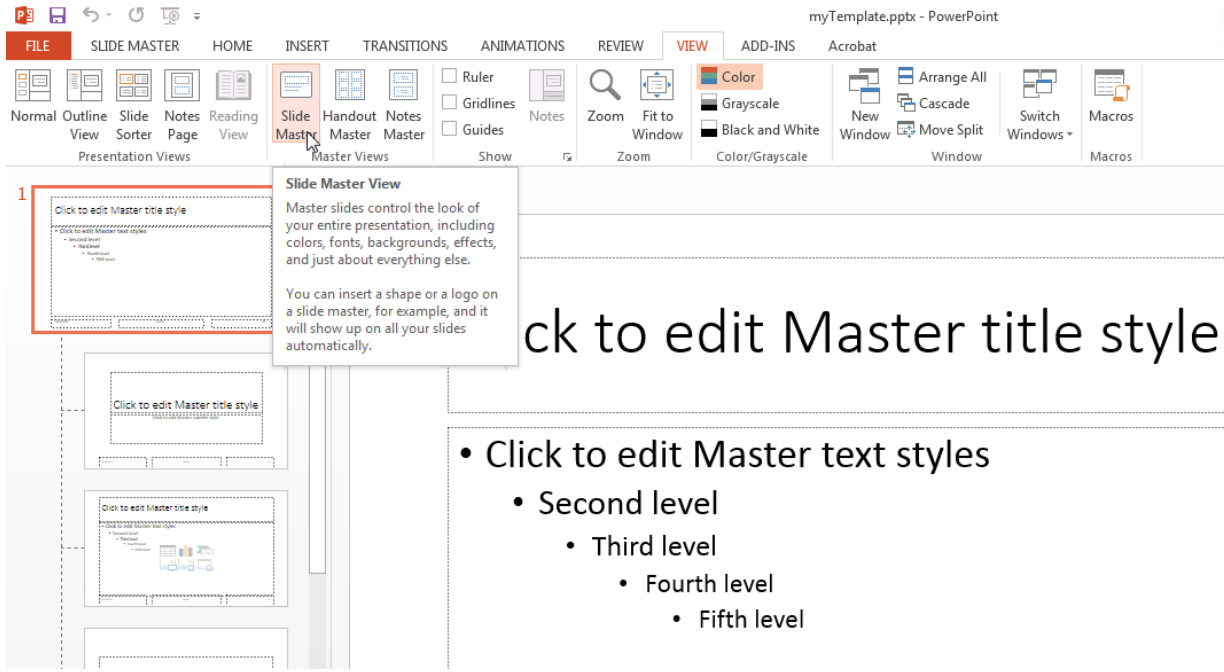
3. Open the presentation and make changes to the template elements.
4. Save the presentation using a different name. Using a different name prevents you from overwriting it with the default template.
5. Use the new template with a PPT API presentation. For example, if the customized template is called `myTemplate`, then use `myTemplate` when you create a PPT API presentation:

```
newPresentation = Presentation('mySecondPresentation','myTemplate');
```

Global Presentation Formatting Using a Slide Master

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

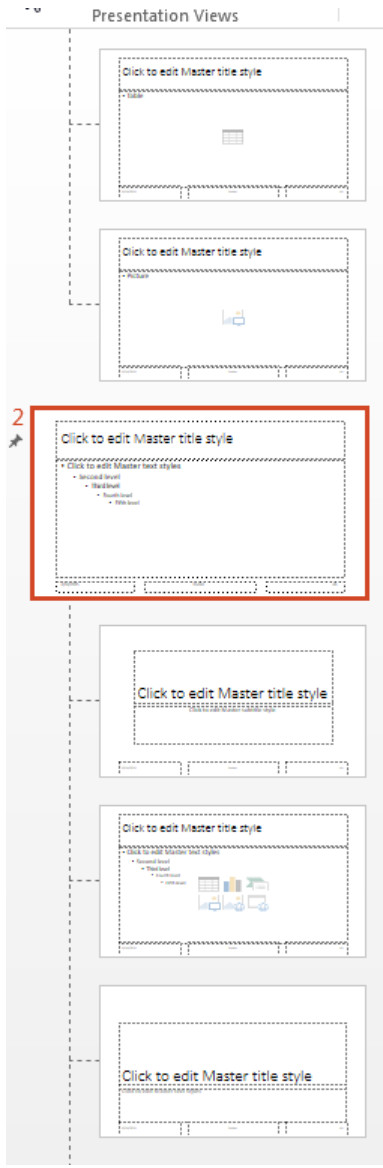
You can add a slide master to a PowerPoint template. Adding a slide master is useful for providing different formatting for different parts of a presentation.

1 Interactively open the PowerPoint template.

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

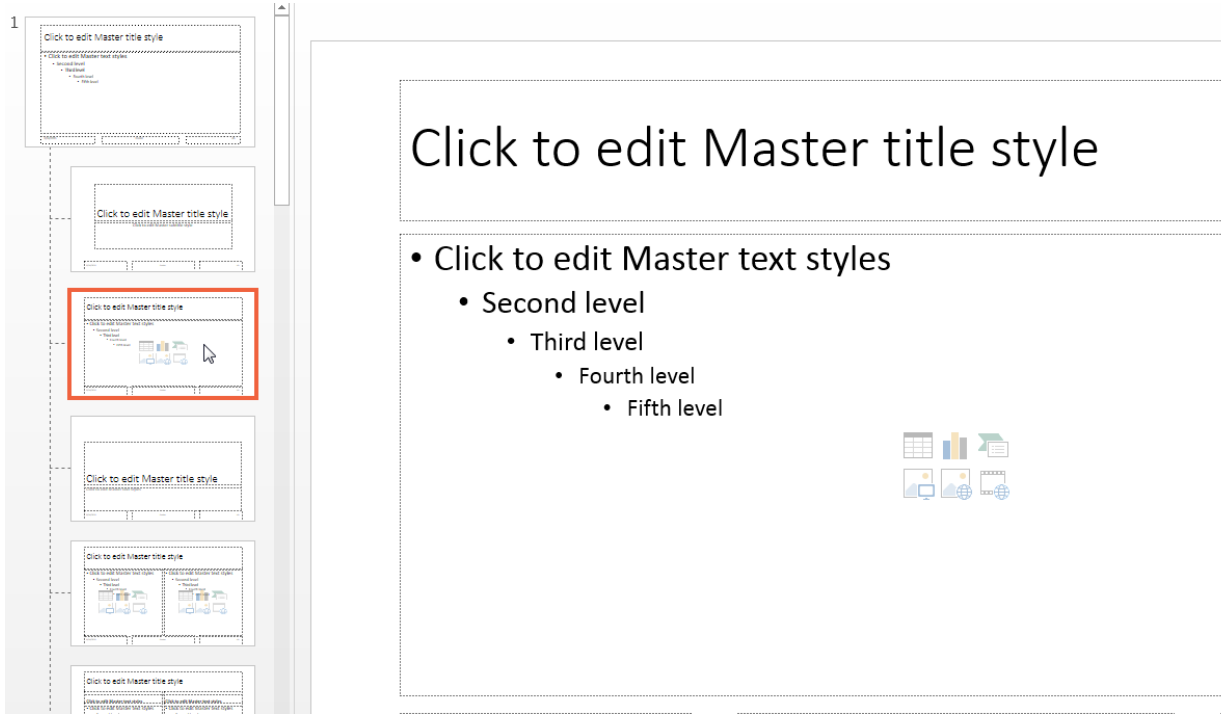
To specify formatting to apply to a specific kind of slide, use a slide layout.

1 In PowerPoint, open a template or a presentation that you want to use as a template.

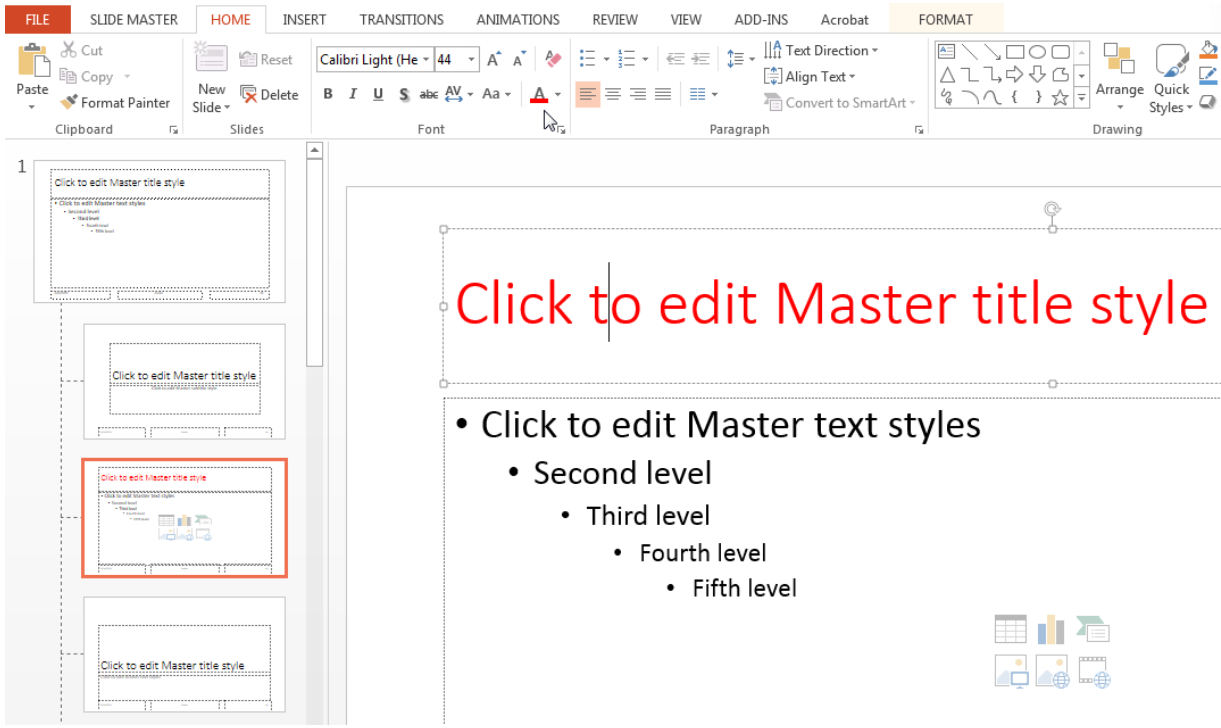
2 In the **View** tab, in the **Master Views** section, click **Slide Master**.

3 From the slide masters and layout pane, select the slide layout whose formatting you want to change. For example, in the default PPT API PowerPoint template, click the `Title and Content` slide layout.

Tip To see the name of a slide layout, hover over that layout. A tooltip appears with the name of the slide layout and the number of slides that use that slide 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.

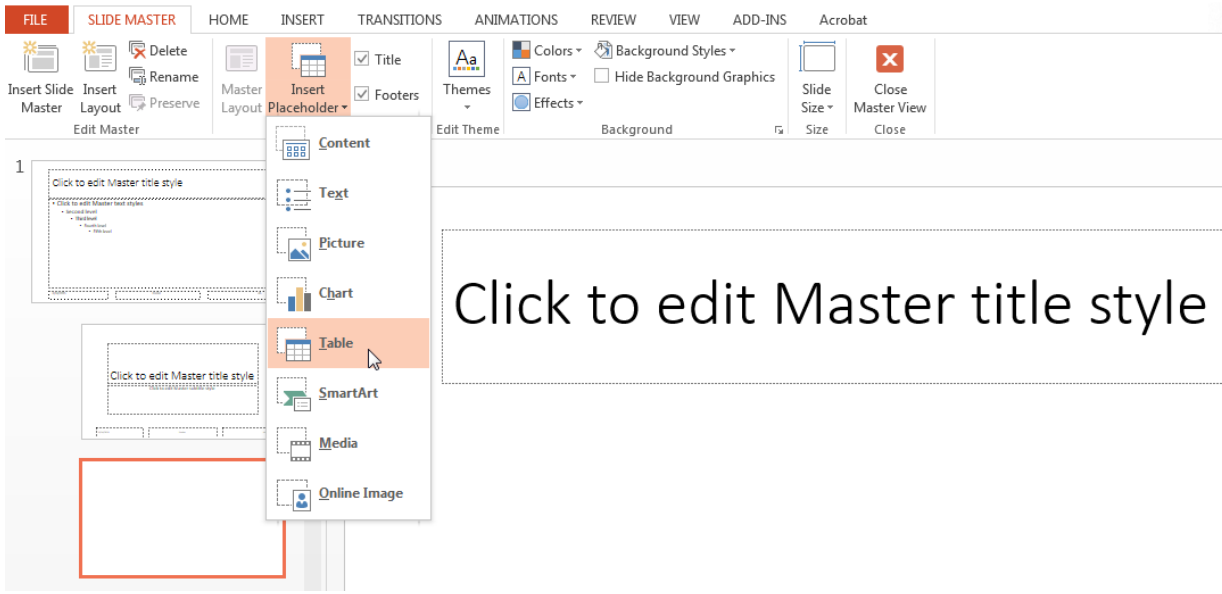
1 Interactively open the PowerPoint template that you want to modify.

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.

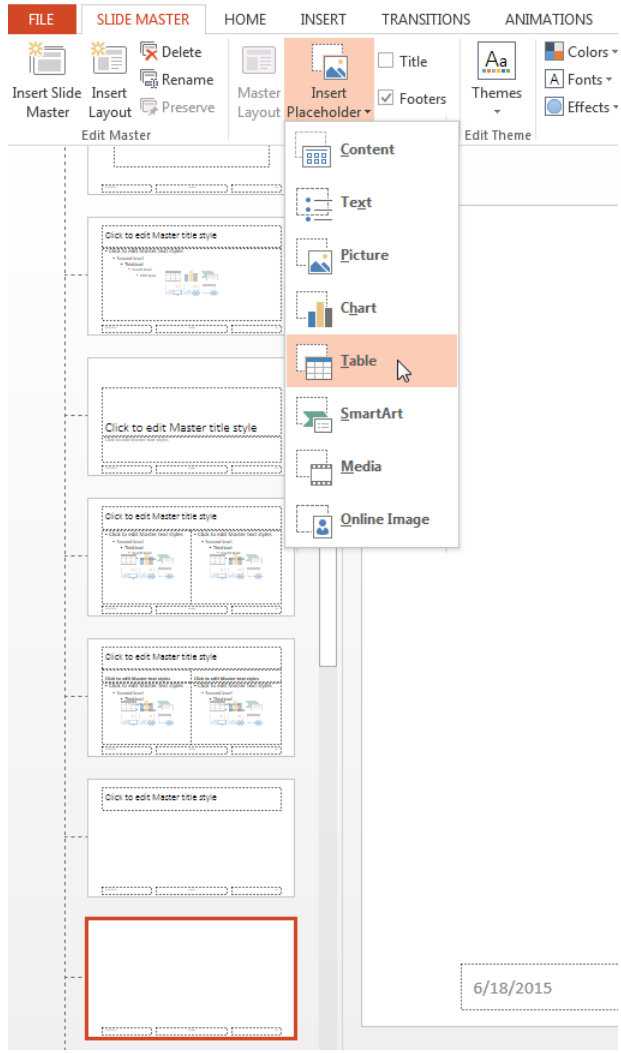
6 Save the template.

Add a Placeholder

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. Interactively open the PowerPoint template that you want to modify.
2. In the **View** tab, in the **Master Views** section, click **Slide Master**.
3. In the slide layout pane, select the slide layout to add the placeholder to.
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 Save the template.

Related Examples

- "Access PowerPoint Template Elements" on page 14-37

More About

- "Presentation Formatting Approaches" on page 14-20

Access PowerPoint Template Elements

In this section...
"PPT API Applications and PowerPoint Templates" on page 14-5
"Template Elements" on page 14-5
"View and Change Slide Master Names" on page 14-38
"View and Change Slide Layout Names" on page 14-39
"View and Change Placeholder and Content Object Names" on page 14-41

PPT API Applications and PowerPoint Templates

The PPT API uses PowerPoint presentations as templates to generate presentations. 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
- An existing PowerPoint presentation whose content you want to update
- A PowerPoint template

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 PPT API comes with a default template that you can use to create presentations. If the default template does not meet your needs, you can use PowerPoint interactively to create templates that do meet your needs.

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

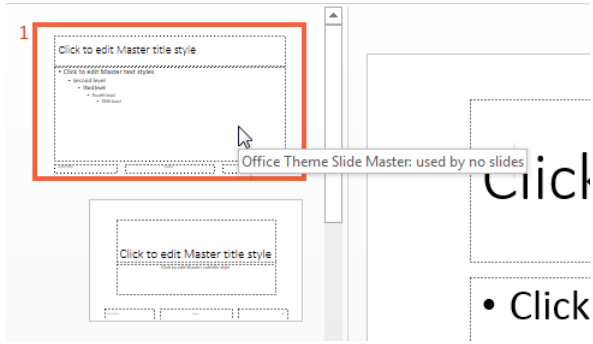
PowerPoint Template Element	Purpose
Slide layouts	Specifies a variant of a slide master layout.
Table styles	Specifies the default appearance of a table. PowerPoint defines a standard set of table styles. You cannot modify these styles but you can use the PPT API to apply these styles to tables you create and override the styles for particular tables.
Placeholders	Specifies an area of a slide layout that you can replace with a text string, 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 string `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 `mlreportgen.ppt.Presentation` object. This example uses the default PPT API PowerPoint template, which has one slide master.

```
import mlreportgen.ppt.*;
slides = Presentation('myPresentation');
getMasterNames(slides);

ans =

    'Office Theme'
```

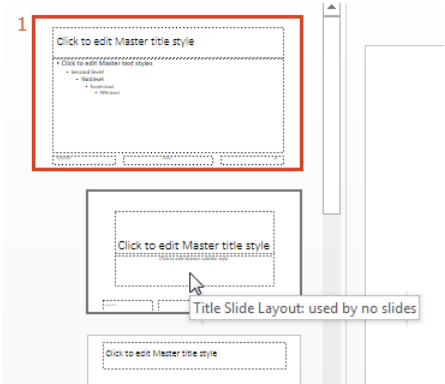
View and Change Slide Layout Names

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 string `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

  Columns 6 through 11

    'Title and Content'  'Section Header'  'Two Content'  'Comparison'  'Title Only'

  Columns 12 through 13

    'Content with Capt…'  'Picture with Capt…'
```

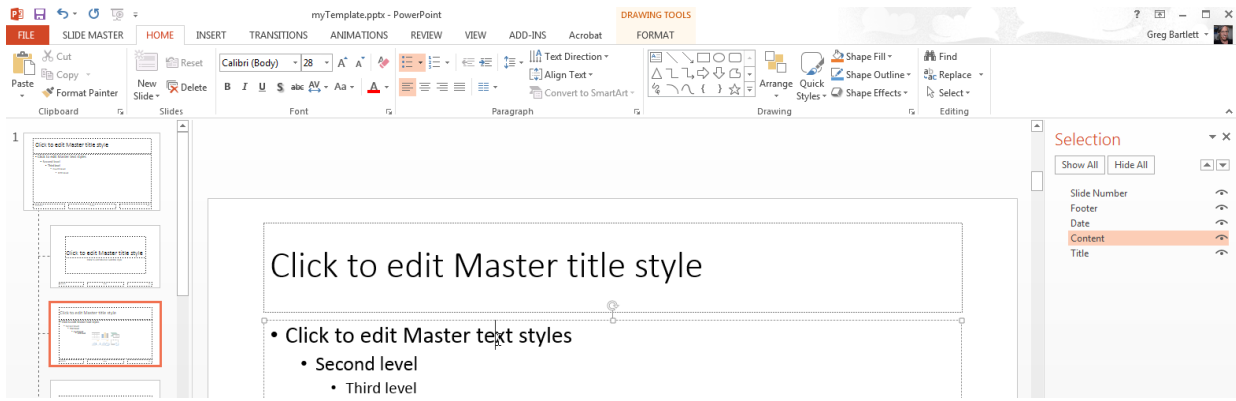
View and Change Placeholder and Content Object Names

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.*

slidesFile = 'myTablePresentation.pptx';
```

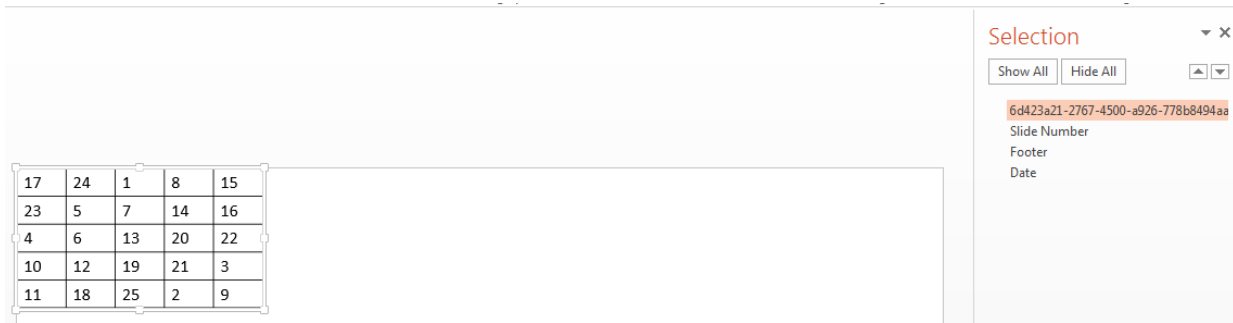
```
slides = Presentation(slidesFile);

slide1 = add(slides,'Blank');
add(slide1,Table(magic(5)));

close(slides);

if ispc
    winopen(slidesFile);
end
```

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.



Related Examples

- “Set Up a PowerPoint Template” on page 14-26

More About

- “Presentation Formatting Approaches” on page 14-20

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.*;

slides = Presentation('myParaPres');

add(slides,'Title and Content');
add(slides,'Title and Content');

caution = {FontColor('red'),Bold(true)};
p1 = Paragraph('Hardware Requirements');
p1.Style = caution;
p2 = Paragraph('Software Requirements');
p2.Style = caution;

titles = find(slides,'Title');

replace(titles(1),p1);
replace(titles(2),p2);
```

```
close(slides);
```

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.

Related Examples

- “Use Format Properties” on page 14-45
- “Define a Style Using Format Objects” on page 14-43
- “Set Up a PowerPoint Template” on page 14-26

More About

- “Presentation Formatting Approaches” on page 14-20

Use Format Properties

In this section...
“Dot Notation” on page 14-45
“Get the Properties of an Object” on page 14-45
“Set the Properties of an Object” on page 14-46

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`.

```
par1 = Paragraph('My paragraph');
```

To specify the `Bold` property for the `para1` object, use:

```
par1.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');
para.Bold
```

```
ans =

     []
```

Set the Properties of an Object

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 `para1` paragraph to bold:

```
par1 = Paragraph('My paragraph');
para1.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)};
```

Related Examples

- “Define a Style Using Format Objects” on page 14-43
- “Set Up a PowerPoint Template” on page 14-26

More About

- “Presentation Formatting Approaches” on page 14-20

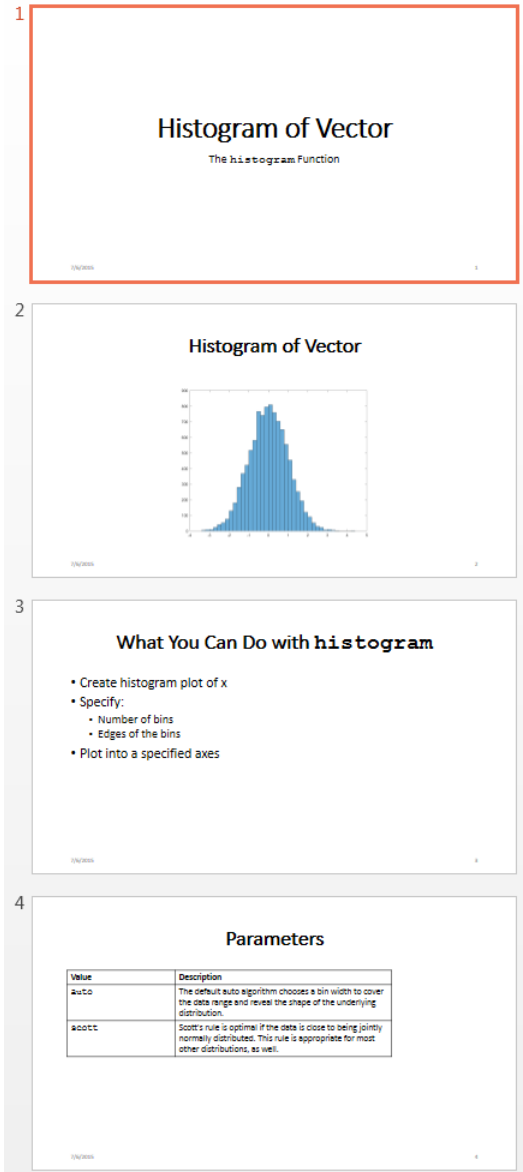
Update Presentation Content Programmatically

In this section...
“Generate the Existing Presentation” on page 14-48
“Updates to the Presentation” on page 14-50
“Set Up the Existing Presentation” on page 14-52
“Import the PPT API Package” on page 14-53
“Create the `Presentation` Object” on page 14-53
“Replace a Picture” on page 14-53
“Replace Text with Links” on page 14-54
“Replace a Table” on page 14-54
“Insert a New Slide” on page 14-55
“Generate and Open the Presentation” on page 14-55
“Code for `myUpdatedPresentation`” on page 14-56

You can use the PPT API to update content programmatically in an existing PowerPoint presentation.

Generate the Existing Presentation

This example updates content in the PowerPoint presentation `myNewPPTPresentation`. Although you create the presentation programmatically, after you generate it, the presentation is like any other PowerPoint presentation. To generate the presentation, click myNewPPTPresentation program and execute the code in MATLAB. The presentation includes four slides:



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 `myNewPPTPresentation` 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

Histogram of Vector

The `histogram` Function

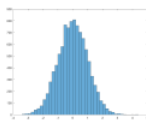
2

Histogram Plots

- You can use the `histogram` function to create many types of plots.

3

Histogram of Vector



4

What You Can Do with `histogram`

- Create histogram plot of `x`
- Specify:
  - Number of bins
  - Edges of the bins
- Plot into a specified axes

5

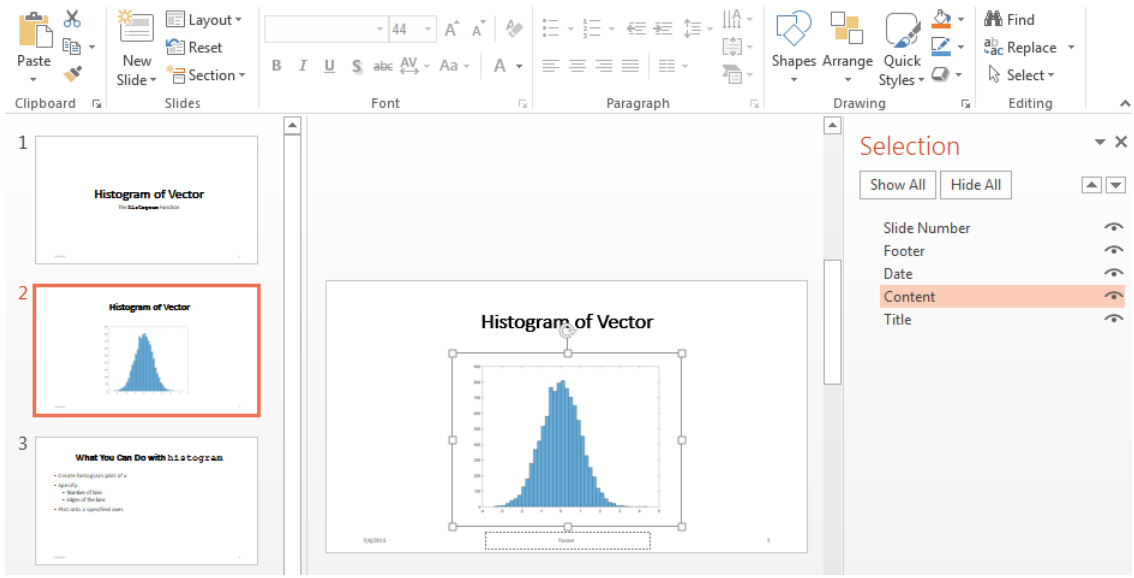
Parameters

Value	Description
`'auto'`	The default auto algorithm chooses a bin width to cover the data range and reveal the shape of the underlying distribution.
`'scott'`	Scott's rule is optimal if the data is close to being jointly normally distributed. This rule is appropriate for most other distributions, as well.

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 `myNewPPTPresentation` 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 `HistBins` and the `Content` object name to `Histogram`.
   - In the third slide, change `Title` to `RelatedFuncs`. Change `Content` to `FuncList`.

- In the fourth slide, change `Content` 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.

Create the `Presentation` Object

Create a `Presentation` object. Specify:

- `myUpdatedPresentation.pptx` as the output file for the generated presentation.
- `myNewPPTPresentation.pptx` as the PowerPoint template. Use the presentation file that you want to update as the template file.

```
slidesFile = 'myUpdatedPresentation.pptx';
slides = Presentation(slidesFile,'myNewPPTPresentation.pptx');
```

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(slides,'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(slides,'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(slides,'RelatedFuncs',funcsTitle);

histCounts = Paragraph();
histCountsLink = ExternalLink...
('http://www.mathworks.com/help/matlab/ref/histcounts.html','histcounts');
append(histCounts,histCountsLink);

fewerbins = Paragraph();
fewerbinsLink = ExternalLink...
('http://www.mathworks.com/help/matlab/ref/fewerbins.html','fewerbins');
append(fewerbins,fewerbinsLink);

replace(slides,'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(slides,'ParamTable');
replace(slides,'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(slides,'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.

```
slides.Children(2).Name = 'ReferenceSlide';

refSlide = find(slides,'ReferenceSlide');
introSlide = add(slides,'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 Open the Presentation

Generate the PowerPoint presentation. Use a `close` method with a `Presentation` object.

```
close(slides);
```

Open the presentation `myUpdatedPresentation.pptx` file. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```

Code for `myUpdatedPresentation`

Here is the complete PPT API program to create the `myUpdatedPresentation` presentation.

Note: This code requires that the `myNewPPTPresentation.pptx` file be in your current folder. To generate that presentation, click myNewPPTPresentation program and execute the code in MATLAB. 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 14-52.

```
import mlreportgen.ppt.*;

slidesFile = 'myUpdatedPresentation.pptx';
slides = Presentation(slidesFile,'myNewPPTPresentation.pptx');

histTitle = Paragraph('Histogram with Specified Bin Edges');
replace(slides,'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(slides,'HistBins',plotEdges)

funcsTitle = Paragraph('Related Functions');
replace(slides,'RelatedFuncs',funcsTitle);

histCounts = Paragraph();
histCountsLink = ExternalLink...
('http://www.mathworks.com/help/matlab/ref/histcounts.html','histcounts');
append(histCounts,histCountsLink);

fewerbins = Paragraph();
```

```
fewerbinsLink = ExternalLink...
('http://www.mathworks.com/help/matlab/ref/fewerbins.html','fewerbins');
append(fewerbins,fewerbinsLink);

replace(slides,'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(slides,'ParamTable');
replace(slides,'ParamTable',table2);


slides.Children(2).Name = 'ReferenceSlide';

refSlide = find(slides,'ReferenceSlide');
introSlide = add(slides,'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(slides);
if ispc
    winopen(slidesFile);
end
```

Related Examples

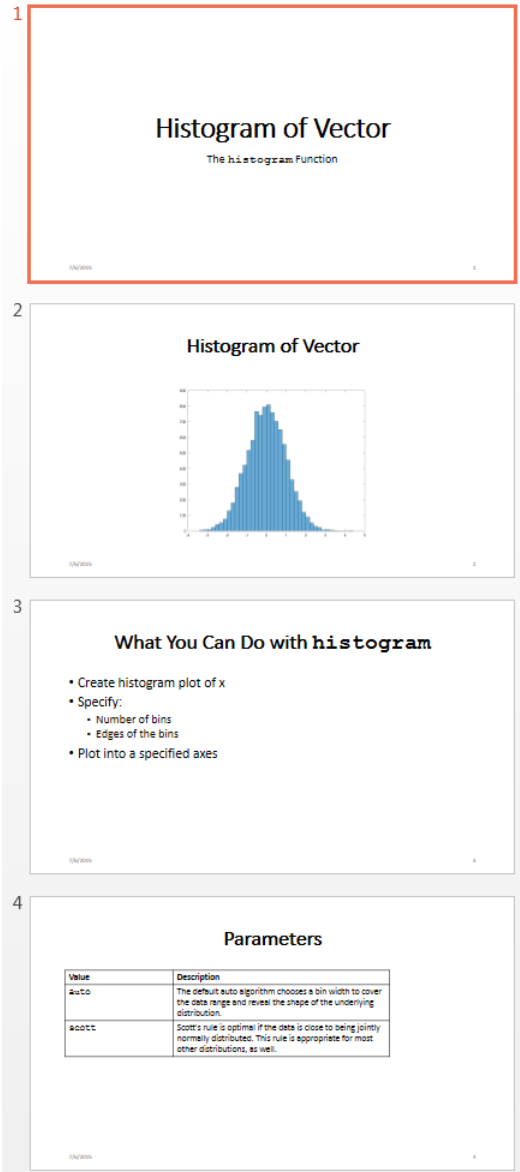
- “Create a Presentation Programmatically” on page 14-59

- "Set Up a PowerPoint Template" on page 14-26
- "Access PowerPoint Template Elements" on page 14-37
- "Add Slides" on page 14-71
- "Create and Format Text" on page 14-81
- "Create and Format Paragraphs" on page 14-84
- "Create and Format Tables" on page 14-87
- "Create and Format Pictures" on page 14-96
- "Create and Format Links" on page 14-98

Create a Presentation Programmatically

In this section...
“Set Up a Template” on page 14-61
“Import the PPT API Package” on page 14-63
“Create the `Presentation` Object” on page 14-63
“Add a Presentation Title Slide” on page 14-64
“Add a Slide with a Picture” on page 14-65
“Add a Slide with Text” on page 14-65
“Add a Slide with a Table” on page 14-66
“Generate and Open the Presentation” on page 14-67
“Code for `myNewPPTPresentation`” on page 14-68

This presentation example shows some common tasks involved in creating a presentation with the PPT API. This example produces these slides:



To use the PPT API to create a complete PowerPoint presentation programmatically, you:

- Set up an empty PowerPoint presentation as a template for the presentation.
- In MATLAB, import the `mlreportgen.ppt` PPT API package.
- Create a `Presentation` object that contains the presentation code.
- Add slides based on slide layouts in the template.
- Add content to the slides.
- Generate the presentation.

Tip To see another example of a PPT API program in MATLAB, enter `population_slides`.

Set Up a Template

A PPT API program uses a PowerPoint presentation as a template to generate a presentation. When you create a complete presentation programmatically, use an empty template. If slides in the template have content (such as text or tables), the content appears in the presentation that the PPT API program generates.

The PPT API provides a default PowerPoint template. You can use the PPT API to make a copy of the default template, which you then can customize to use with your PPT API program. This code creates a template called `myTemplate`, which is a copy of the default PPT API template.

```
import mlreportgen.ppt.*
slidesFile = 'myTemplate.pptx';
slides = Presentation('myTemplate');

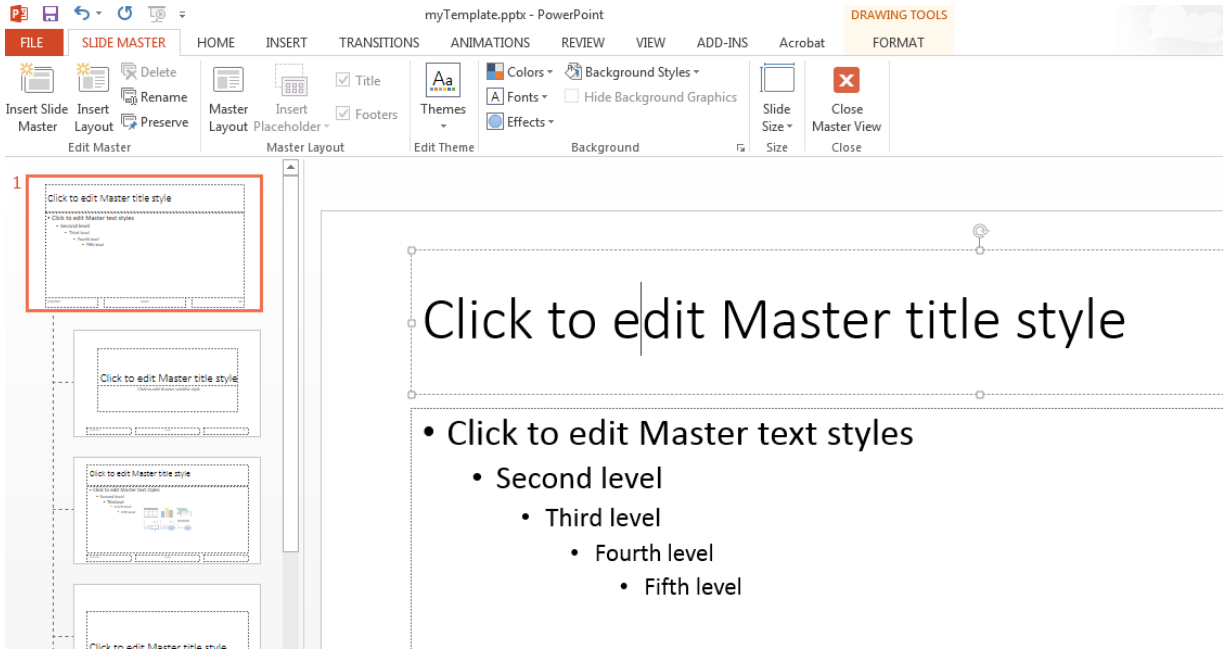
open(slides);

close(slides);
```

Open the `myTemplate.pptx` file. On a Windows platform, you can open the presentation in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```

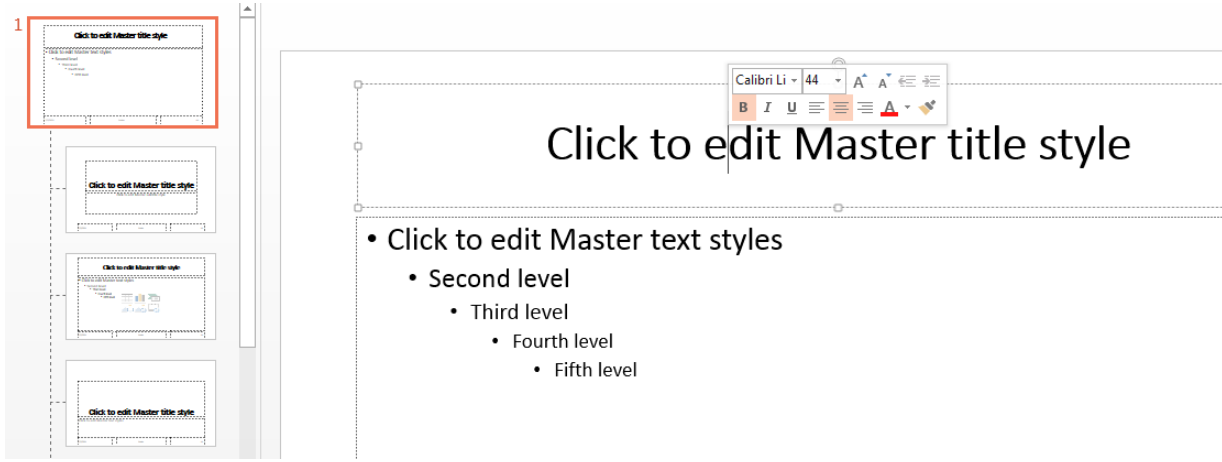
To see template elements, such as the slide master and slide layouts, in PowerPoint **View** pane, click **Slide Master**.



Use PowerPoint interactively to customize the template. To set default formatting for the whole presentation, customize a slide master. To set default formatting for a specific kind of slide, customize a slide layout. For example, you can use the slide master to set up the template to use bold text for slide titles.

1 In the slide layout, right-click in `Click to edit Master title style` box.

2 From the context menu, select **B** (bold). Also select the button to center the text.



3 Save and close the template.

Import the PPT API Package

All PPT API class names include the prefix `mlreportgen.ppt`. To avoid including the package name when you invoke PPT API object constructors and method, import the package. Insert this statement at the beginning of a PPT API program.

```
import mlreportgen.ppt.*;
```

Note: The `import` line is the first line in this example program. This example creates a PPT API program in sections, and so you use the `import` command only once. To view the complete program, click myNewPPTPresentation program.

Create the `Presentation` Object

Create a `Presentation` object. Specify:

- `myNewPPTPresentation.pptx` as the output file for the generated presentation.
- `myTemplate.pptx` as the PowerPoint template.

```
slidesFile = 'myNewPPTPresentation.pptx';
slides = Presentation(slidesFile,'myTemplate');
```

Add a Presentation Title Slide

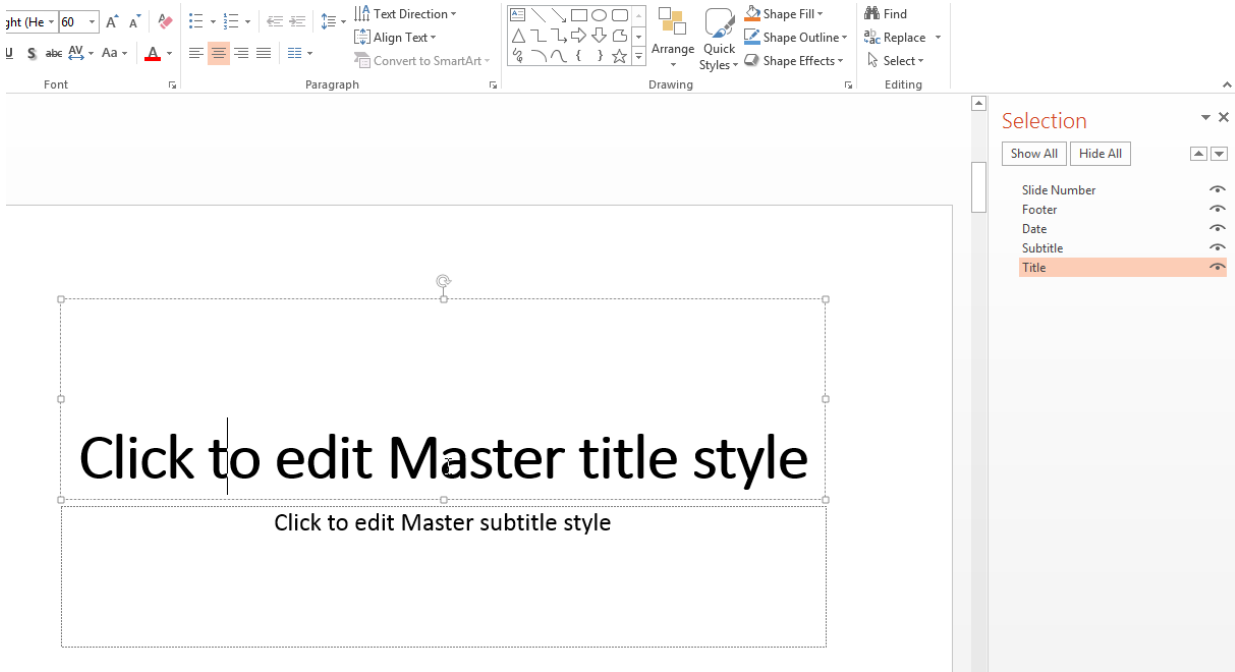
To add a slide programmatically, specify a slide layout in the template. To see the names of the slide layouts, in the PowerPoint **Slide Master** tab, hover over a slide layout.

The `myTemplate` template includes a `Title Slide` slide layout for the presentation title slide. To add a slide using the `Title Slide` layout, use the `add` method with `slides`, which is a `Presentation` object. In the slide layout name, do not include the word `Layout`, which appears at the end of slide layout names when you hover over slide layouts.

```
presentationTitleSlide = add(slides,'Title Slide');
```

To add content to the slide, first find out the names of the content objects in the slide layout.

1. In PowerPoint, stay in the slide master view and select the **Home** tab.
2. Click **Select** > **Selection Pane**.
3. In the slide layout, click the slide layout content item whose name you want.



Specify a title and a subtitle. Specify the slide, the name of the content objects you want to replace, and the text for the title and subtitle. For the subtitle, to specify a different font for the word `histogram`, use a `Paragraph` object for that text.

```
replace(presentationTitleSlide,'Title','Create Histogram Plots');

subtitleText = Paragraph('The ');
funcName = Text('histogram');
funcName.Font = 'Courier New';

append(subtitleText,funcName);
append(subtitleText,' Function');
replace(presentationTitleSlide,'Subtitle1',subtitleText);
```

Add a Slide with a Picture

To add a picture to a slide, create a `Picture` object, specifying an image file. This example creates a MATLAB plot and saves the plot as an image file. You can add the picture to a slide. Use a `Title and Content` slide layout and add a title and picture.

```
x = randn(10000,1);
h = histogram(x);

saveas(gcf,'myPlot_img.png');

plot1 = Picture('myPlot_img.png');

pictureSlide = add(slides,'Title and Content');
replace(slides,'Title','Histogram of Vector');
contents = find(pictureSlide,'Content');
replace(contents(1),plot1);
```

Add a Slide with Text

Depending on the slide layout, PowerPoint formats the text you add as a paragraph, a bulleted list, or a numbered list. This example creates another instance of a `Title and Content` slide, which formats the text as a bulleted list. You can use a nested cell array to specify levels for bullets.

```
textSlide = add(slides,'Title and Content');

titleText2 = Paragraph('What You Can Do with ');
func = Text('histogram');
```

```
func.Font = 'Courier New';
append(titleText2,func);
contents = find(textSlide,'Title');
replace(contents(1),titleText2);

contents = find(textSlide,'Content');
replace(contents(1),{'Create histogram plot of x',...
'Specify:',{'Number of bins','Edges of the bins'},...
'Plot into a specified axes'});
```

Add a Slide with a Table

You can use several approaches to add a table to a slide. This example shows how to build a table row by row.

- Create a `Table` object.
- Create a `TableRow` object for each row of the table.
- Create `TableEntry` objects and append them to table rows.
- Add the table to a slide.

```
tableSlide = add(slides,'Title and Content');
contents = find(tableSlide,'Title');
titleText3 = Paragraph('Parameters');
replace(contents(1),titleText3);

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 cover ');
append(tr2te2Text,'the data range and reveal the shape of the underlying 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);

contents = find(tableSlide,'Content');
replace(contents(1),paramTable);
```

Generate and Open the Presentation

Generate the PowerPoint presentation. Use a `close` method with a `Presentation` object.

```
close(slides);
```

Open `myNewPPTPresentation.pptx`. On a Windows platform, you can open it in MATLAB:

```
if ispc
    winopen(slidesFile);
end
```

Code for `myNewPPTPresentation`

Here is the complete PPT API program to create `myNewPPTPresentation`.

Note: The `myTemplate.pptx` file must be in the current folder. If it is not, see "Set Up a Template" on page 14-61.

```
import mlreportgen.ppt.*;

slidesFile = 'myNewPPTPresentation.pptx';
slides = Presentation(slidesFile,'myTemplate');

%Add a title slide
presentationTitleSlide = add(slides,'Title Slide');
replace(presentationTitleSlide,'Title','Create Histograms Plots');

subtitleText = Paragraph('The ');
funcName = Text('histogram');
funcName.Font = 'Courier New';
>> append(subtitleText,funcName);
append(subtitleText,' Function');
replace(presentationTitleSlide,'Subtitle',subtitleText);

%Add a picture slide
x = randn(10000,1);
h = histogram(x);

saveas(gcf,'myPlot_img.png');

plot1 = Picture('myPlot_img.png');

pictureSlide = add(slides,'Title and Content');
replace(slides,'Title','Histogram of Vector');
contents = find(pictureSlide,'Content');
replace(contents(1),plot1);

%Add a text slide
textSlide = add(slides,'Title and Content');

titleText2 = Paragraph('What You Can Do with ');
func = Text('histogram');
func.Font = 'Courier New';
```

```
append(titleText2,func);
contents = find(textSlide,'Title');
replace(contents(1),titleText2);

contents = find(textSlide,'Content');
replace(contents(1),{'Create histogram plot of x',...
'Specify:',{'Number of bins','Edges of the bins'},...
'Plot into a specified axes'});

%Add a table slide
tableSlide = add(slides,'Title and Content');
contents = find(tableSlide,'Title');
titleText3 = Paragraph('Parameters');
replace(contents(1),titleText3);

paramTable = Table();
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 underlying distribu
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('Scott''s rule 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);

contents = find(tableSlide,'Content');
replace(contents(1),paramTable);

%Generate and open the presentation
close(slides);

if ispc
    winopen(slidesFile);
end
```

Related Examples

- “Update Presentation Content Programmatically” on page 14-48
- “Set Up a PowerPoint Template” on page 14-26
- “Access PowerPoint Template Elements” on page 14-37
- “Add Slides” on page 14-71
- “Create and Format Text” on page 14-81
- “Create and Format Paragraphs” on page 14-84
- “Create and Format Tables” on page 14-87
- “Create and Format Pictures” on page 14-96
- “Create and Format Links” on page 14-98

Add Slides

In this section...
“Specify the Order of a Slide” on page 14-71
“Specify the Slide Master” on page 14-73

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 14-32.

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.*;
slides = Presentation('myPresentation');
slide1 = add(slides,'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.

```
slides = Presentation('myPresentation');
titleSlide = add(slides,'Title Slide');
contentSlide = add(slides,'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`.

```
slides = Presentation('myPresentation');
titleSlide = add(slides,'Title Slide');
introSlide = add(slides,'Title Slide');
pictureSlide = add(slides,'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.

   ```
   slides = Presentation('myPresentation','myPresentation');
   open(slides);

   slides.Children(2).Name = 'ReferenceSlide';
   close(slides);
   ```
3. To identify the reference slide object, use the slide name. Add the new slide relative to the reference slide.

   ```
   slides = Presentation('myPresentation', 'myPresentation');
   open(slides);

   refSlide = find(slides, 'ReferenceSlide');
   add(slides, 'Blank', refSlide);

   close(slides);
   ```

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.

```
slides = Presentation('myPresentation');

titleSlide = add(slides,'Title Slide');
introSlide = add(slides,'Title and Content');
```

```
pictureSlide = add(slides,'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.

```
slides = Presentation('myPresentation');
titleSlide = add(slides,'Title Slide',myCustomMaster);
```

See Also

Functions

mlreportgen.ppt.Presentation.add | mlreportgen.ppt.Presentation.getLayoutNames | mlreportgen.ppt.Presentation.getMasterNames

Related Examples

- “Create a Presentation Object to Hold Content” on page 14-13
- “Add and Replace Presentation Content” on page 14-74

Add and Replace Presentation Content

In this section...
“Set Up the Template” on page 14-74
“Replace Content” on page 14-75
“Add and Replace Text” on page 14-75
“Add or Replace a Table” on page 14-78
“Add or Replace a Picture” on page 14-79

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 Template" on page 14-26
- "Access PowerPoint Template Elements" on page 14-37

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(slides,'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.

```
slides = Presentation('myPresentation');
titleSlide = add(slides,'Title Slide');

contents = find(slides,'Title');
replace(contents(1),'This Is My Title');
```

Add and Replace Text

You can use these approaches to add or replace text for to a presentation.

Text Specification Technique	Associated PPT API Objects
Specify a string as part of creating these objects.	• `Text` • `Paragraph` • `ExternalLInk`

Text Specification Technique	Associated PPT API Objects
Append text to a paragraph or external link.	Append text to these PPT API objects: • `Paragraph` • `TableEntry` • `ExternalLink`
Replace a `Paragraph` object in a presentation or slide.	Specify a string, `Paragraph` object, or a cell array of strings or `Paragraph` objects or a combination of both kinds of objects, for the `replace` method with these objects: • `Presentation` • `Slide`
Add to or replace text in a placeholder object.	• Add to a `ContentPlaceholder` object a string, `Paragraph` object, or with a cell array of strings or `Paragraph` objects, or a combination of both. • Replace a `ContentPlaceholder` object with a `Paragraph` object. • Add to a `TextBoxPlaceholder` object a string, `Paragraph` object, or with a cell array of strings or `Paragraph` objects or a combination of both. • Replace a `TextBoxPlaceholder` object with a `Paragraph` object. See "Add and Replace Text in Placeholders" on page 14-76.
Add to, or replace, a text box.	Add to or replace a `TextBox` object with a string, `Paragraph` object, or with a cell array of strings or `Paragraph` objects, or a combination of both. See "Add or Replace Text in a Text Box" on page 14-77.

Add and Replace Text in Placeholders

You can add or replace text in a `ContentPlaceholder` and a `TextBoxPlaceholder`, specifying:

- A string
- A `Paragraph` object
- A cell array of strings or `Paragraph` objects or a combination of strings 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';
slides = Presentation(name1);
add(slides,'Comparison');
replace(slides, 'Left Content', 'dummy content');
replace(slides, 'Right Content', 'dummy content');
close(slides);

name2 = 'after';
slides = Presentation(name2, name1);

lefts = find(slides, 'Left Content');
rights = find(slides, '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(slides);

if ispc
    winopen(slides.OutputPath);
end
```

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.*;
slides = Presentation('myPresentation.pptx');
titleSlide = add(slides,'Title Slide');

tb = TextBox();
tb.X = '2in';
tb.Y = '2in';
tb.Width = '5in';
add(tb,'Text for text box');

add(titleSlide,tb);
close(slides);
```

Add or Replace a Table

To add or replace a table in a presentation, first create a `Table` object. You can add a table by using an `add` method with a `Slide` object.

```
import mlreportgen.ppt.*;
slides = Presentation('myPresentation.pptx');
tableSlide = add(slides,'Blank');

magicTable = Table(magic(5));
magicTable.X = '3in';
MagicTable.Y = '5in';

add(tableSlide,magicTable);

close(slides);
```

To replace content in a table, replace the whole table. To replace a `Table` object, use the `replace` method with the `Table` object, and specify another `Table` object. To replace a table in a slide content placeholder, use the `replace` method with the `Slide` object and specify a `Table` object.

```
slides = Presentation('myPresentation');
```

```
tableSlide = add(slides,'Title and Table');

table1 = Table(magic(9));
contents = find(tableSlide,'Table');
replace(contents(1),table1);

close(slides);
```

Add or Replace a Picture

You can add a picture by using an `add` method or a `replace` method with a `Slide` object.

```
import mlreportgen.ppt.*;
slides = Presentation('myPresentation.pptx');
pictureSlide = add(slides,'Blank');

plane = Picture(which('b747.jpg'));
plane.X = '2in';
plane.Y = '2in';
plane.Width = '5in';
plane.Height = '2in';

add(pictureSlide,plane);

close(slides);
```

You can use a `replace` method with a `Picture` or `PicturePlaceholder` object. For example, the default template has a `Title and Picture` slide layout that has a picture placeholder called `Picture`.

```
import mlreportgen.ppt.*;
slides = Presentation('myPresentation');
pictSlide = add(slides,'Title and Picture');

plane2 = Picture(which('b747.jpg'));
contents = find(pictSlide,'Picture');
replace(contents(1),plane2);

close(slides);
```

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.

Related Examples

- "Create and Format Text" on page 14-81
- "Create and Format Paragraphs" on page 14-84
- "Create and Format Tables" on page 14-87
- "Create and Format Pictures" on page 14-96
- "Create and Format Links" on page 14-98

More About

- "Presentation Formatting Approaches" on page 14-20

Create and Format Text

In this section...
“Create Text” on page 14-81
“Create a Subscript or Superscript” on page 14-81
“Format Text” on page 14-82

Create Text

You can create a `Text` object using an `mlreportgen.ppt.Text` constructor, specifying a text string.

Also, you can create text by using a string with these kinds of PPT API objects:

- `Paragraph`
- `ExternalLink`
- `TableEntry`
- `TextBox`
- `ContentPlaceholder`
- `TextBoxPlaceholder`

For example:

```
import mlreportgen.ppt.*;
slides = Presentation('myPresentation.pptx');
slide1 = add(slides,'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 14-75.

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.ExternalLink | mlreportgen.ppt.Paragraph | mlreportgen.ppt.Text | mlreportgen.ppt.TextBox

Related Examples

- “Add and Replace Text” on page 14-75

More About

- "Presentation Formatting Approaches" on page 14-20

Create and Format Paragraphs

In this section...
“Create a Paragraph” on page 14-84 “Format Paragraph Content” on page 14-84

Create a Paragraph

To create a `Paragraph` object, use the `mlreportgen.ppt.Paragraph` constructor. You can:

- Create an empty `Paragraph` object.
- Specify a string for the paragraph text.
- Specify a `Text` or `ExternalLink` object as the paragraph text.

After you create a `Paragraph` object, you can append strings or `Text` objects to add text to the paragraph. You can specify separate formatting for each `Text` and `ExternalLink` 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 strings that you add. The paragraph formatting applies to `Text` 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);
```

`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`
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

mlreportgen.ppt.ExternalLink | mlreportgen.ppt.Paragraph | mlreportgen.ppt.Text | mlreportgen.ppt.TextBox

Classes

mlreportgen.ppt.TableEntry

Related Examples

- “Add and Replace Text” on page 14-75

More About

- “Presentation Formatting Approaches” on page 14-20

Create and Format Tables

In this section...
“Create a Table” on page 14-87
“Format a Table” on page 14-87
“View Table Style Names” on page 14-93

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, strings, 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(slides);

%% 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(slides);
```

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
slides = Presentation();
add(slides,'Title and Content');
add(slides,'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(slides,'Content');
tbl = replace(contents(1),Table(magic(5)));
tbl.StyleName = 'Medium Style 2 - Accent 1'

%% Replace the second content placholder 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(slides);

if ispc
    winopen(slides.OutputPath);
end
```

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.

`Table` Object Formatting	Format Object	Format Property
Table style from template	n/a	`StyleName`

`Table` Object Formatting	Format Object	Format Property
Use the PowerPoint template to specify table style formatting. Create an instance of the style in your template.		
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.

`TableRow` Object Formatting	Format Object	Format Property
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

`TableRow` Object Formatting	Format Object	Format Property
Text color	`FontColor`	n/a
Bold	`Bold`	n/a
Italic	`Italic`	n/a
Strike	`Strike`	n/a
Underline	`Underline`	n/a
Background color	`BackgroundColor`	n/a

To specify default formatting for the contents of a `TableEntry` object, use these formatting options.

`TableEntry` Object Formatting	Format Object	Format Property
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.*
slidesFile = 'myColSpecs.pptx'
slides = Presentation(slidesFile);
add(slides,'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(slides,'Content',t);

close(slides);
if ispc
winopen(slides.OutputPath);
end
```

When you create a `ColSpec` object, you can specify the column width in the constructor. For example:

```
myColSpec = ColSpec('3in');
```

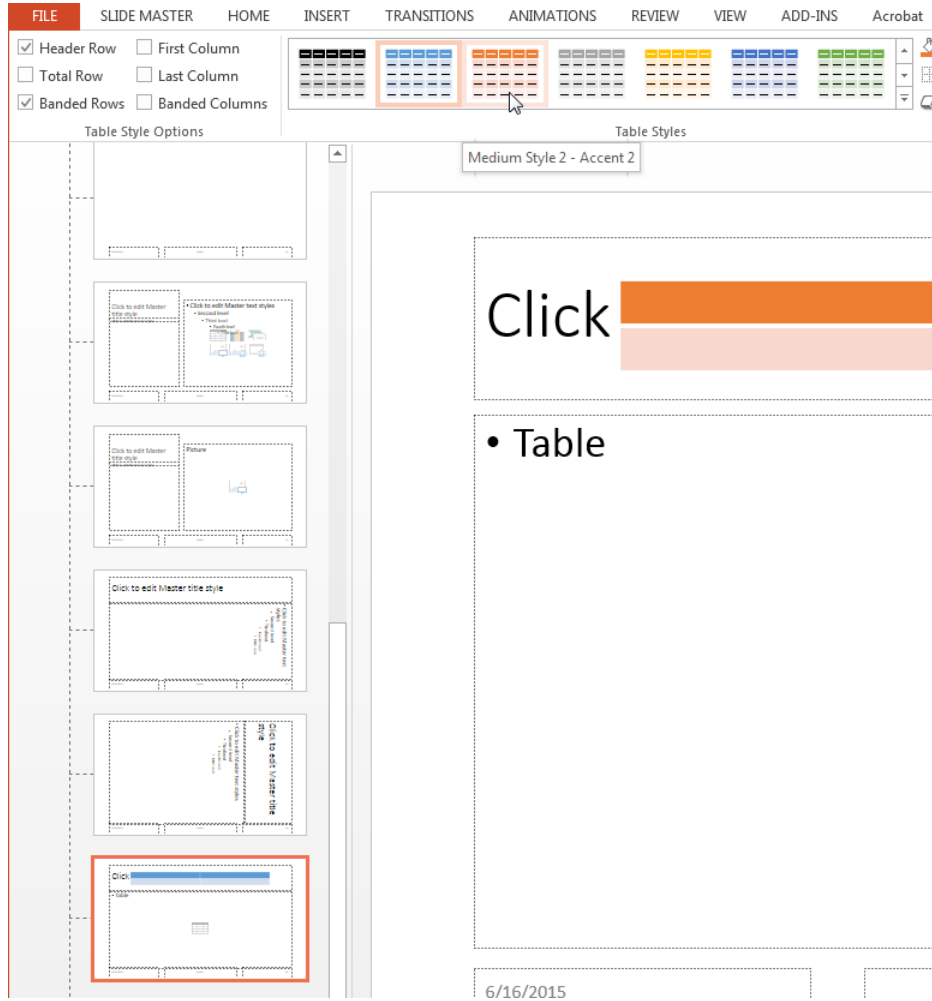
Also, you can specify the column width using the Width property of a `ColSpec` object. You specify other formatting properties of the `ColSpec` object, such as `BackgroundColor`.

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 `mlreportgen.ppt.Presentation` object. The output in this example shows just the first two of many table styles in the default template.

```
import mlreportgen.ppt.*
slides = Presentation('myPlaceholderPresentation');

getTableStyleNames(slides)
```

```
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 string or the numeric identifier string.

See Also

Functions

mlreportgen.ppt.Table.entry | mlreportgen.ppt.Table.row

Classes

mlreportgen.ppt.ColSpec | mlreportgen.ppt.ColWidth | mlreportgen.ppt.Table | mlreportgen.ppt.TableEntry | mlreportgen.ppt.TablePlaceholder | mlreportgen.ppt.TableRow

Related Examples

- “Add or Replace a Table” on page 14-78

More About

- “Presentation Formatting Approaches” on page 14-20

Create and Format Pictures

In this section...
“Create a Picture” on page 14-96 “Format a Picture” on page 14-97

Create a Picture

To create a picture for a presentation, use the `mlreportgen.ppt.Picture` constructor. Specify the path to a picture file. The picture file must use one of these formats (you cannot use `.svg` format):

- `.bmp`
- `.emf`
- `.eps`
- `.gif`
- `.jpeg`
- `.jpg`
- `.png`
- `.tif`
- `.tiff`

For example:

```
import mlreportgen.ppt.*
slides = Presentation('slides');
pictureSlide = add(slides,'Blank');

plane = Picture(which('b747.jpg'));
plane.Width = '5in';
plane.Height = '2in';

add(pictureSlide,plane);

close(slides);
```

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 14-79

More About

- “Presentation Formatting Approaches” on page 14-20

Create and Format Links

In this section...
“Create an External Link” on page 14-98
“Format an External Link” on page 14-98

Create an External Link

To create a link to a location outside of a presentation, use the `mlreportgen.ppt.ExternalLink` constructor. Specify the full URL of the link target and specify the link text as a string. For example:

```
import mlreportgen.ppt.*

slidesFile = 'myExternalLinkPresentation.pptx';
slides = Presentation(slidesFile);

add(slides,'Title and Content');

p = Paragraph('This is a link to the ');
link = ExternalLink('http://www.mathworks.com','MathWorks site.');

append(p,link);
replace(slides,'Content',p);

close(slides);

if ispc
    winopen(slidesFile);
end
```

Format an External Link

To specify default formatting for the link text, use the `Style` property with format objects.

`ExternalLink` Object Formatting	Format Object	Format Property
Font family	`FontFamily`	n/a

`ExternalLink` Object Formatting	Format Object	Format Property
Font family for complex scripts to handle locales	`FontFamily`	n/a
Text color	`FontColor`	n/a
Font size	`FontSize`	n/a
Bold	`Bold`	n/a
Italic	`Italic`	n/a
Strike	`Strike`	n/a
Underline	`Underline`	n/a
Background color	`BackgroundColor`	n/a

See Also

Classes

mlreportgen.ppt.ExternalLink

More About

- “Presentation Formatting Approaches” on page 14-20

15

Classes Being Removed

mlreportgen.dom.DOCXRawFormat class

Package: mlreportgen.dom

XML markup for array of Microsoft Word formats

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Markup` — Word XML markup strings

cell array of strings

Specify a cell array of strings. Each string contains Word XML markup for a Word format.

For information about XML markup for Word formats, see http://officeopenxml.com/anatomyofOOXML.php.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

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

More About

- “Report Formatting Approaches” on page 13-21

mlreportgen.dom.DOCXPageSize class

Package: mlreportgen.dom

Size and orientation of pages in Microsoft Word document

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) | string

String specifying the height of the page. The string 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`width` — Width of page

`'8.5in'` (default) | string

The string 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`orientation` — Orientation of page

string

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

string

String specifying the page height. The string 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`Id` — ID for document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Orientation` — Orientation (portrait or landscape) for pages in section

string

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 document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Width` — Width of page

`'8.5in'` (default) | string

The string 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
- `pi` — 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');
```

- “Report Formatting Approaches” on page 13-21

See Also

mlreportgen.dom.DOCXPageMargins | mlreportgen.dom.DOCXSection

mlreportgen.dom.DOCXPageMargins class

Package: mlreportgen.dom

Page margins for Microsoft Word page layout

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
string

String specifying the width of the bottom margin. The string 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
- `pi` — picas
- `pt` — points
- `px` — pixels

`Footer` — Footer size
string

Specify the size using the same format used for the `Bottom` property.

`Gutter` — Gutter size
string

Specify the size using the same format used for the `Bottom` property.

`Header` — Header size
string

Specify the size using the same format used for the `Bottom` property.

`Id` — ID for document element
string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated ID.

`Left` — Left margin size
string

Specify the size using the same format used for the `Bottom` property.

`Right` — Right margin size
string

Specify the size using the same format used for the `Bottom` property.

`Tag` — Tag for document element
string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

`Top` — Top margin size

string

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

More About

- “Report Formatting Approaches” on page 13-21

mlreportgen.dom.DOCXSection class

Package: mlreportgen.dom

Page format and layout for Microsoft Word document section

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 document element

string

A session-unique ID is generated as part of document element creation. You can specify an ID to replace the generated 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 `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

`PageMargins` — Margin sizes and page orientation in this section

`mlreportgen.dom.DOCXPageMargins` object

Margin sizes and page orientation in this section, specified as an `mlreportgen.dom.DOCXPageMargins` object.

`PageSize` — Size of pages in this section

`mlreportgen.dom.DOCXPageSize` object

Size of pages in this section, specified as an `mlreportgen.dom.DOCXPageSize` object.

`Parent` — Parent of document element

DOM object

This read-only property lists the parent of this document element.

`RawFormats` — XML markup for unsupported section formats

cell array

Cell array of strings, with each string containing Word XML markup for a Word format. For information about XML markup for Word formats, see http://officeopenxml.com/anatomyofOOXML.php.

`Style` — Formats defining section style

array of format objects

The formats you specify using this property override corresponding formats defined by the stylesheet style specified by the `StyleName` property. The DOM interface ignores formats that do not apply to this element.

`Tag` — Tag for document element

string

Tag for document element, specified as a string.

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.

For example, to make it easier to identify where an issue occurred during document generation, you can specify your own tag value.

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.DOCXPageFooter | mlreportgen.dom.DOCXPageHeader | mlreportgen.dom.DOCXPageMargins | mlreportgen.dom.DOCXPageSize | mlreportgen.dom.DOCXRawFormat | mlreportgen.dom.DOCXSubDoc

More About

- “Report Formatting Approaches” on page 13-21