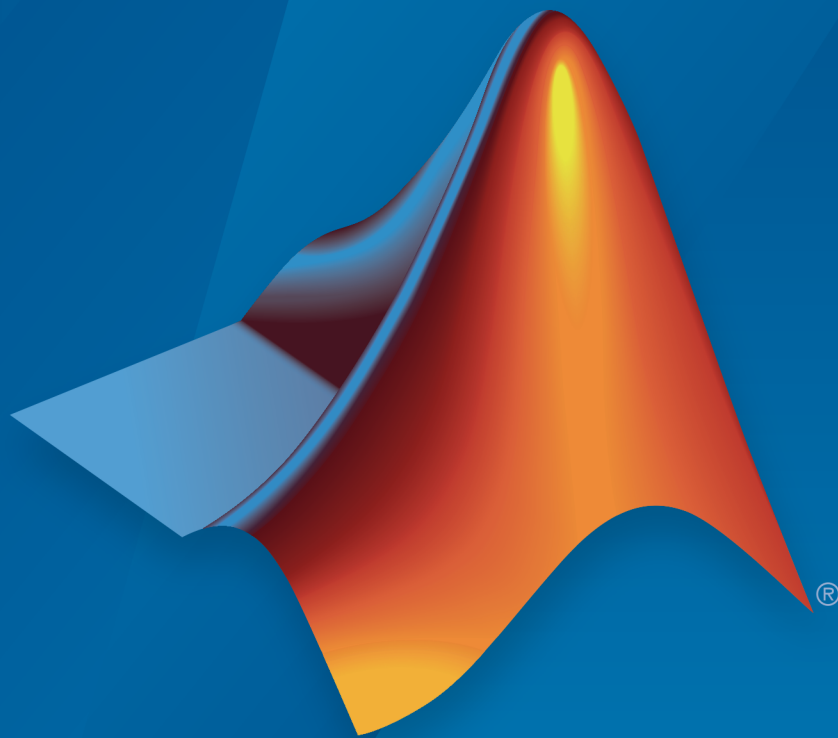


MATLAB[®] Production Server[™]

Code Deployment



MATLAB[®]

R2015a



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[®] Production Server[™] Code Deployment

© COPYRIGHT 2012–2015 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

March 2014	Online only	New for Version 1.2 (Release R2014a)
October 2014	Online only	Revised for Version 2.0 (Release R2014b)
March 2015	Online only	Revised for Version 2.1 (Release R2015a)

Write Deployable MATLAB Code

1

Deployment Coding Guidelines	1-2
State-Dependent Functions	1-3
Does My MATLAB Function Carry State?	1-3
Defensive Coding Practices	1-3
Techniques for Preserving State	1-4
Deploying MATLAB Functions Containing MEX Files	1-6
Unsupported MATLAB Data Types for Client and Server Marshaling	1-7

Create a Deployable Archive from MATLAB Code

2

Compile Deployable Archives with Production Server Compiler App	2-2
Compile Deployable Archives from Command Line	2-5
Execute Compiler Projects with deploytool	2-5
Compile a Deployable Archive with mcc	2-5
Modifying Deployed Functions	2-7

Customize the Installer	3-2
Change the Application Icon	3-2
Add Application Information	3-3
Change the Splash Screen	3-3
Change the Installation Path	3-4
Change the Logo	3-4
Edit the Installation Notes	3-5
Manage Required Files in Compiler Project	3-6
Dependency Analysis	3-6
Using the Compiler Apps	3-6
Using mcc	3-6
Specify Files to Install with Application	3-8
Manage Support Packages	3-9
Using a Compiler App	3-9
Using the Command Line	3-10

Advanced Uses of the Command Line Compiler

Simplify Compilation Using Macros	4-2
Macros	4-2
Working With Macros	4-2
Invoke MATLAB Build Options	4-4
Specify Full Path Names to Build MATLAB Code	4-4
Using Bundles to Build MATLAB Code	4-5
MATLAB Runtime Component Cache and Deployable Archive	
Embedding	4-7
Overriding Default Behavior	4-8
For More Information	4-8

5

6

Write Deployable MATLAB Code

- “Deployment Coding Guidelines ” on page 1-2
- “State-Dependent Functions” on page 1-3
- “Deploying MATLAB Functions Containing MEX Files” on page 1-6
- “Unsupported MATLAB Data Types for Client and Server Marshaling” on page 1-7

Deployment Coding Guidelines

MATLAB coding guidelines are essentially the same for both the deployment products and MATLAB Production Server with important distinctions regarding functions that depend on MATLAB state.

Functions you deploy with MATLAB Production Server cannot be assumed to retain access to the same instance of the MATLAB Runtime, since the workers can access a number of different MATLAB Runtime instances. Therefore, when using MATLAB Production Server you must take extra care to ensure that state has not been changed or invalidated. See “State-Dependent Functions” on page 1-3 for more information.

Refer to “Write Deployable MATLAB Code” for general guidelines about deploying MATLAB code.

State-Dependent Functions

MATLAB code that you want to deploy often carries *state*—a specific data value in a program or program variable.

Does My MATLAB Function Carry State?

Example of carrying state in a MATLAB program include, but are not limited to:

- Modifying or relying on the MATLAB path and the Java[®] class path
- Accessing MATLAB state that is inherently persistent or global. Some example of this include:
 - Random number seeds
 - Handle Graphics[®] root objects that retain data
 - MATLAB or MATLAB toolbox settings and preferences
- Creating global and persistent variables.
- Loading MATLAB objects (MATLAB classes) into MATLAB. If you access a MATLAB object in any way, it loads into MATLAB.
- Calling MEX files, Java methods, or C# methods containing static variables.

Defensive Coding Practices

If your MATLAB function not only carries state, but *relies on it* for your function to properly execute, you must take additional steps (listed in this section) to ensure state retention.

When you deploy your application, consider cases where you carry state, and safeguard against that state's corruption if needed. *Assume* that your state may be changed and code defensively against that condition.

The following are examples of “defensive coding” practices:

Reset System-Generated Values in the Deployed Application

If you are using a random number seed, for example, reset it in your deployed application program to ensure the integrity of your original MATLAB function.

Validate Global or Persistent Variable Values

If you must use global or persistent variables, always validate their value in your deployed application and reset if needed.

Ensure Access to Data Caches

If your function relies on cached replies to previous requests, for instance, ensure your deployed system and application has access to that cache outside of the MATLAB environment.

Use Simple Data Types When Possible

Simple data types are usually not tied to a specific application and means of storing state. Your options for choosing an appropriate state-preserving tool increase as your data types become less complicated and specific.

Avoid Using MATLAB Callback Functions

Avoid using MATLAB callbacks, such as `timer`. Callback functions have the ability to interrupt and override the current state of the MATLAB Production Server worker and may yield unpredictable results in multiuser environments.

Techniques for Preserving State

The most appropriate method for preserving state depends largely on the type of data you need to save.

- Databases provide the most versatile and scalable means for retaining stateful data. The database acts as a generic repository and can generally work with any application in an enterprise development environment. It does not impose requirements or restrictions on the data structure or layout. Another related technique is to use comma-delimited files, in applications such as Microsoft[®] Excel[®].
- Data that is specific to a third-party programming language, such as Java and C#, can be retained using a number of techniques. Consult the online documentation for the appropriate third-party vendor for best practices on preserving state.

Caution Using MATLAB `LOAD` and `SAVE` functions is often used to preserve state in MATLAB applications and workspaces. While this may be successful in some

circumstances, it is highly recommended that the data be validated and reset if needed, if not stored in a generic repository such as a database.

Deploying MATLAB Functions Containing MEX Files

If the MATLAB function you are deploying uses MEX files, ensure that the system running MATLAB Production Server is running the version of MATLAB Compiler™ used to create the MEX files.

Coordinate with your server administrator and application developer as needed.

Unsupported MATLAB Data Types for Client and Server Marshaling

These data types are not supported for marshaling between MATLAB Production Server server instances and clients:

- MATLAB function handles
- Complex (imaginary) data
- Sparse arrays

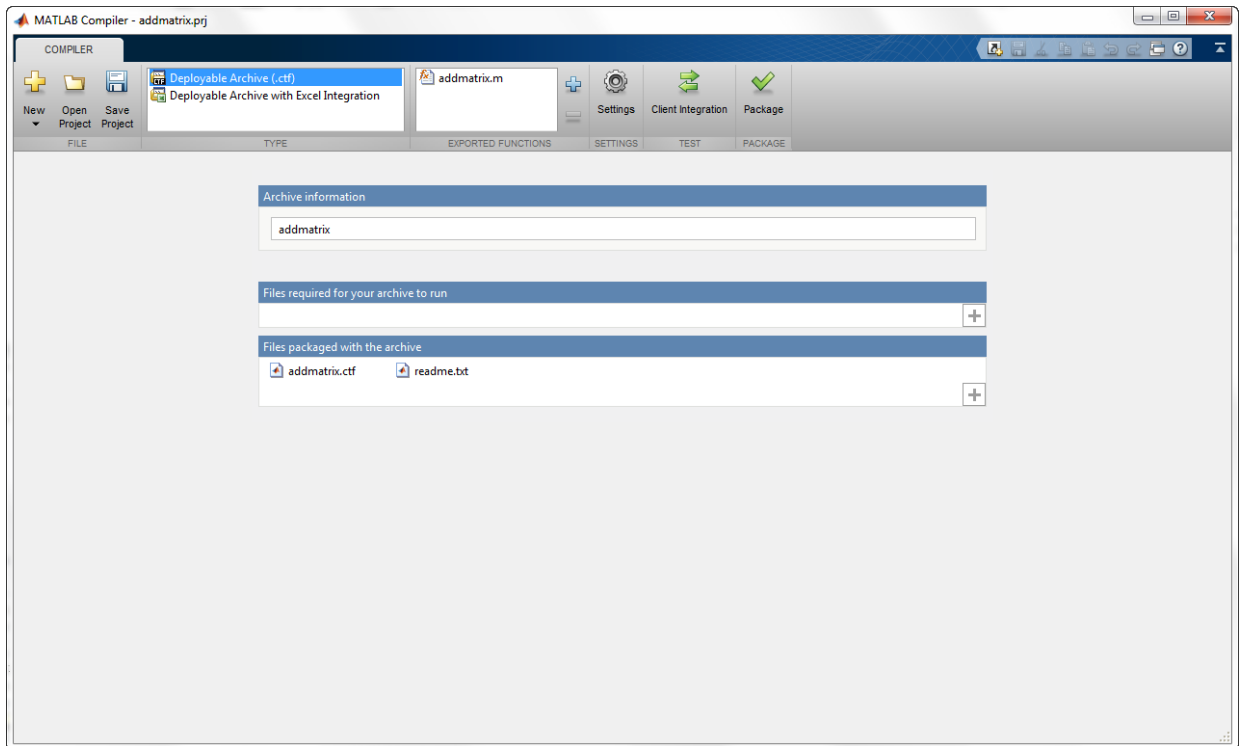
Create a Deployable Archive from MATLAB Code

- “Compile Deployable Archives with Production Server Compiler App” on page 2-2
- “Compile Deployable Archives from Command Line” on page 2-5
- “Modifying Deployed Functions” on page 2-7

Compile Deployable Archives with Production Server Compiler App

To compile MATLAB code into a deployable archive:

- 1 Open the Production Server Compiler app.
 - a On the toolstrip select the **Apps** tab on the toolstrip.
 - b Click the arrow at the far right of the tab to open the apps gallery.
 - c Click **Production Server Compiler**.



Note: To open an existing project, select it from the MATLAB **Current Folder** panel.

Note: You can also launch the Production Server Compiler app using the `productionServerCompiler` function.

- 2 In the **Application Type** section of the toolstrip, select **Deployable Archive**.

Note: If the **Application Type** section of the toolstrip is collapsed, you can expand it by clicking the down arrow.

- 3 Specify the MATLAB files you want deployed in the package.
 - a In the **Exported Functions** section of the toolstrip, click the plus button.

Note: If the **Exported Functions** section of the toolstrip is collapsed, you can expand it by clicking the down arrow.

- b In the file explorer that opens, locate and select one or more the MATLAB files.
 - c Click **Open** to select the file and close the file explorer.

The names of the selected files are added to the list and a minus button appears below the plus button. The name of the first file listed is used as the default application name.

- 4 In the **Archive Information** section of the app, specify the name of the archive.
- 5 In the **Files required for your application to run** section of the app, verify that all of the files required by the deployed MATLAB functions are listed.

Note: These files are compiled into the generated binaries along with the exported files.

The built-in dependency checker will automatically populate this section with the appropriate files. However, if needed you can manually add any files it missed.

For more information see “Manage Required Files in Compiler Project” on page 3-6.

- 6 In the **Files packaged with your archive** section of the app, verify that any additional non-MATLAB files you want packaged with the archive are listed.

Note: These files are placed in the `applications` folder of the installation.

This section automatically lists:

- Generated deployable archive
- Readme file

You can manually add files to the list. Additional files can include documentation, sample data files, and examples to accompany the application.

For more information see “Specify Files to Install with Application” on page 3-8.

- 7** Click the **Settings** button to customize the flags passed to the compiler and the folders to which the generated files are written.
- 8** Click the **Package** button to compile the MATLAB code and generate any installers.
- 9** Verify that the generated output contains:
 - `for_redistribution` — A folder containing the installer to distribute the archive
 - `for_testing` — A folder containing the raw generated files to create the installer
 - `PackagingLog.txt` — A log file generated by the compiler

Compile Deployable Archives from Command Line

In this section...

“Execute Compiler Projects with `deploytool`” on page 2-5

“Compile a Deployable Archive with `mcc`” on page 2-5

You can compile deployable archives from both the MATLAB command line and the system terminal command line:

- `deploytool` invokes one of the compiler apps to execute a presaved compiler project
- `mcc` invokes the command line compiler

Execute Compiler Projects with `deploytool`

The `deploytool` command has two flags to invoke one of the compiler apps without opening a window.

- `-build project_name` — Invoke the correct compiler app to build the project and not generate an installer.
- `-package project_name` — Invoke the correct compiler app to build the project and generate an installer.

For example, `deploytool -package magicssquare` generates the binary files defined by the `magicssquare` project and packages them into an installer that you can distribute to others.

Compile a Deployable Archive with `mcc`

The `mcc` command invokes the raw compiler and provides fine-level control over the compilation of the deployable archive. It, however, cannot package the results in an installer.

To invoke the compiler to generate a deployable archive use the `-W CTF:component_name` flag with `mcc`. The `-W CTF:component_name` flag creates a deployable archive called `component_name.ctf`.

For compiling deployable archives, you can also use the following options.

Compiler Options

Option	Description
<code>-a filePath</code>	Add any files on the path to the generated binary.
<code>-d outFolder</code>	Specify the folder into which the results of compilation are written.
<code>class{className:mfilename...}</code>	Specify that an additional class is generated that includes methods for the listed MATLAB files.

Modifying Deployed Functions

Once you have built a deployable archive, you can modify your MATLAB code, recompile, and see the change instantly reflected in the archive hosted on your server. This is known as “hot deploying” or “redeploying” a function.

To Hot Deploy, you must have a server created and running, with the built deployable archive located in the server’s `auto_deploy` folder.

The server deploys the updated version of your archive when on the following occurs:

- Compiled archive has an updated time stamp
- Change has occurred to the archive contents (new file or deleted file)

It takes a maximum of five seconds to redeploy a function using Hot Deployment. It takes a maximum of ten seconds to undeploy a function (remove the function from being hosted).

Customizing a Compiler Project

- “Customize the Installer” on page 3-2
- “Manage Required Files in Compiler Project” on page 3-6
- “Specify Files to Install with Application” on page 3-8
- “Manage Support Packages” on page 3-9

Customize the Installer

In this section...

- “Change the Application Icon” on page 3-2
- “Add Application Information” on page 3-3
- “Change the Splash Screen” on page 3-3
- “Change the Installation Path” on page 3-4
- “Change the Logo” on page 3-4
- “Edit the Installation Notes” on page 3-5

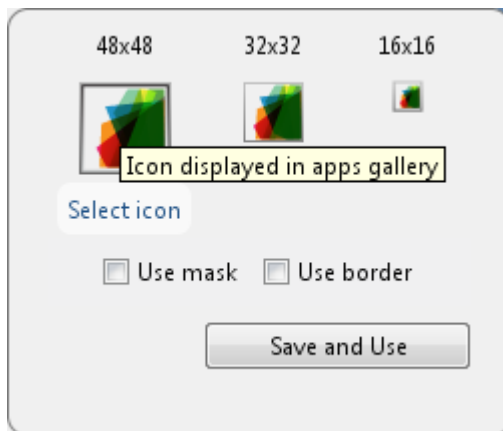
Change the Application Icon

The application icon is used for the generated installer. For standalone applications, it is also the application's icon.

You can change the default icon in **Application Information**. To set a custom icon:

- 1 Click the graphic to the left of the **Application name** field.

A window previewing the icon opens.



- 2 Click **Select icon**.
- 3 Using the file explorer, locate the graphic file to use as the application icon.
- 4 Select the graphic file.

- 5 Click **OK** to return to the icon preview.
- 6 Select **Use mask** to fill any blank spaces around the icon with white.
- 7 Select **Use border** to add a border around the icon.
- 8 Click **Save and Use** to return to the main window.

Add Application Information

The **Application Information** section of the app allows you to provide these values:

- Name

Determines the name of the installed MATLAB artifacts. For example, if the name is `foo`, the installed executable would be `foo.exe`, the Windows[®] start menu entry would be `foo`. The folder created for the application would be *InstallRoot/foo*.

The default value is the name of the first function listed in the **Main File(s)** field of the app.

- Version

The default value is 1.0.

- Author name
- Support e-mail address
- Company name

Determines the full installation path for the installed MATLAB artifacts. For example, if the company name is `bar`, the full installation path would be *InstallRoot/bar/ApplicationName*.

- Summary
- Description

This information is all optional and, unless otherwise stated, is only used for display purposes. It appears on the first page of the installer. On Windows systems, this information is also displayed in the Windows **Add/Remove Programs** control panel.

Change the Splash Screen

The installer's splash screen displays after the installer is started. It is displayed, along with a status bar, while the installer initializes.

You can change the default image by clicking the **Select custom splash screen** link in **Application Information**. When the file explorer opens, locate and select a new image.

Note: You can drag and drop a custom image onto the default splash screen.

Change the Installation Path

Default Installation Paths lists the default path the installer will use when installing the compiled binaries onto a target system.

Default Installation Paths

Windows	C:\Program Files \companyName\appName
Mac OS X	/Applications/companyName/appName
Linux®	/usr/companyName/appName

You can change the default installation path by editing the **Default installation folder** field under **Additional Installer Options**.

The **Default installation folder** field has two parts:

- root folder — A drop down list that offers options for where the install folder is installed. Custom Installation Roots lists the optional root folders for each platform.

Custom Installation Roots

Windows	C:\Users\userName\AppData
Linux	/usr/local

- install folder — A text field specifying the path appended to the root folder.

Change the Logo

The logo displays after the installer is started. It is displayed on the right side of the installer.

You change the default image by clicking the **Select custom logo** link in **Additional Installer Options**. When the file explorer opens, locate and select a new image.

Note: You can drag and drop a custom image onto the default logo.

Edit the Installation Notes

Installation notes are displayed once the installer has successfully installed the packaged files on the target system. They can provide useful information concerning any additional set up that is required to use the installed binaries or simply provide instructions for how to run the application.

The field for editing the installation notes is in **Additional Installer Options**.

Manage Required Files in Compiler Project

In this section...

“Dependency Analysis” on page 3-6

“Using the Compiler Apps” on page 3-6

“Using mcc” on page 3-6

Dependency Analysis

The compiler uses a dependency analysis function to automatically determine what additional MATLAB files are required for the application to compile and run. These files are automatically compiled into the generated binary. The compiler does not generate any wrapper code allowing direct access to the functions defined by the required files.

Using the Compiler Apps

If you are using one of the compiler apps, the required files discovered by the dependency analysis function are listed in the **Files required by your application to run** field.

To add files:

- 1 Click the plus button in the field.
- 2 Select the desired file from the file explorer.
- 3 Click **OK**.

To remove files:

- 1 Select the desired file.
- 2 Press the **Delete** key.

Caution Removing files from the list of required files may cause your application to not compile or to not run properly when deployed.

Using mcc

If you are using `mcc` to compile your MATLAB code, the compiler does not display a list of required files before running. Instead, it compiles all of the required files that are

discovered by the dependency analysis function and adds them to the generated binary file.

You can add files to the list by passing one, or more, `-a` arguments to `mcc`. The `-a` arguments add the specified files to the list of files to be added into the generated binary. For example, `-a hello.m` adds the file `hello.m` to the list of required files and `-a ./foo` adds all of the files in `foo`, and its subfolders, to the list of required files.

Specify Files to Install with Application

The compiler apps package files to install along with the ones it generates. By default the installer includes a readme file with instructions on installing the MATLAB Runtime and configuring it.

These files are listed in the **Files installed with your application** section of the app.

to add files to the list:

- 1 Click the plus button in the field.
- 2 Select the desired file from the file explorer.
- 3 Click **OK** to close the file explorer.

To remove files from the list:

- 1 Select the desired file.
- 2 Press the **Delete** key.

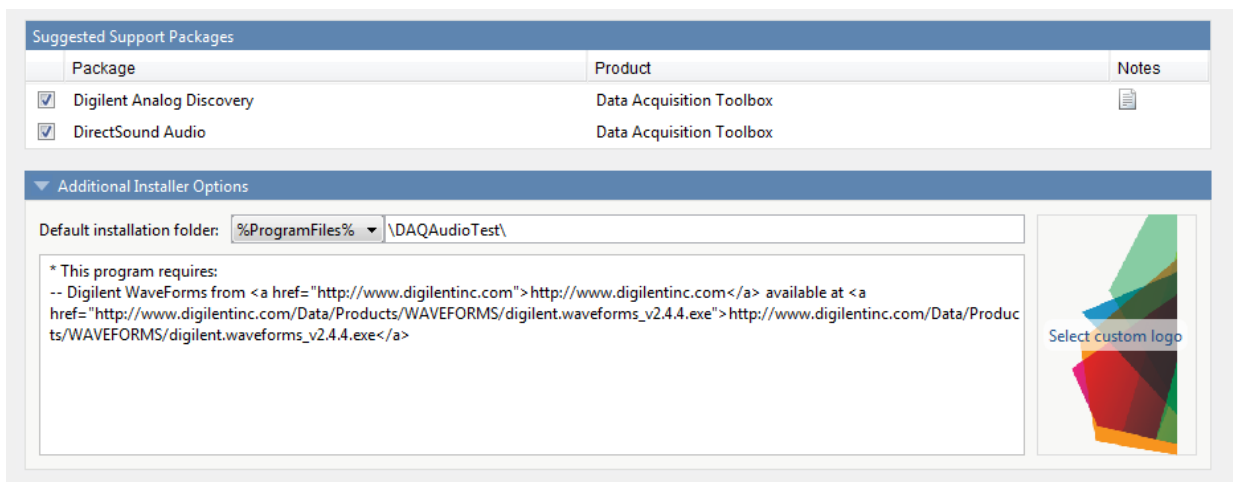
Caution Removing the binary targets from the list results in an installer that does not install the intended functionality.

When installed on a target computer, the files listed in the **Files installed with your application** are placed in the **application** folder.

Manage Support Packages

Using a Compiler App

Many MATLAB toolboxes use support packages to interact with hardware or to provide additional processing capabilities. If your MATLAB code uses a toolbox with an installed support package, the app displays a **Suggested Support Packages** section.



The list displays all installed support packages that your MATLAB code requires. The list is determined using these criteria:

- the support package is installed
- your code has a direct dependency on the support package
- your code is dependent on the base product of the support package
- your code is dependent on at least one of the files listed as a dependency in the `mcc.xml` file of the support package, and the base product of the support package is MATLAB

Deselect support packages that are not required by your application.

Some support packages require third-party drivers that the compiler cannot package. In this case, the compiler adds the information to the installation notes. You can edit

installation notes in the **Additional Installer Options** section of the app. To remove the installation note text, deselect the support package with the third-party dependency.

Caution Any text you enter beneath the generated text will be lost if you deselect the support package.

Using the Command Line

Many MATLAB toolboxes use support packages to interact with hardware or to provide additional processing capabilities. If your MATLAB code uses a toolbox with an installed support package, you pass a `-a` flag to `mcc` when compiling your MATLAB code.

For example, if your function uses the `OS Generic Video Interface` support package.

```
mcc -m -v test.m -a C:\MATLAB\SupportPackages\R2014a\genericvideo
```

Some support packages require third-party drivers that the compiler cannot package. In this case, you are responsible for downloading and installing the required drivers.

Advanced Uses of the Command Line Compiler

- “Simplify Compilation Using Macros” on page 4-2
- “Invoke MATLAB Build Options” on page 4-4
- “MATLAB Runtime Component Cache and Deployable Archive Embedding” on page 4-7

Simplify Compilation Using Macros

In this section...

“Macros” on page 4-2

“Working With Macros” on page 4-2

Macros

The compiler, through its exhaustive set of options, gives you access to the tools you need to do your job. If you want a simplified approach to compilation, you can use one simple *macro* that allows you to quickly accomplish basic compilation tasks. Macros let you group several options together to perform a particular type of compilation.

This table shows the relationship between the macro approach to accomplish a standard compilation and the multioption alternative.

Macro	Bundle	Creates	Option Equivalence
			Function Wrapper Output Stage
-l	macro_option_l	Library	-W lib -T link:lib
-m	macro_option_m	Standalone application	-Wmain-Tlink:exe

Working With Macros

The `-m` option tells the compiler to produce a standalone application. The `-m` macro is equivalent to the series of options

```
-W main -T link:exe
```

This table shows the options that compose the `-m` macro and the information that they provide to the compiler.

-m Macro

Option	Function
-W main	Produce a wrapper file suitable for a standalone application.
-T link:exe	Create an executable link as the output.

Changing Macros

You can change the meaning of a macro by editing the corresponding `macro_option` file in `matlabroot\toolbox\compiler\bundles`. For example, to change the `-m` macro, edit the file `macro_option_m` in the `bundles` folder.

Note This changes the meaning of `-m` for all users of this MATLAB installation.

Specifying Default Macros

As the `MCCSTARTUP` functionality has been replaced by bundle technology, the `macro_default` file that resides in `toolbox\compiler\bundles` can be used to specify default options to the compiler.

For example, adding `-mv` to the `macro_default` file causes the command:

```
mcc foo.m  
to execute as though it were:
```

```
mcc -mv foo.m
```

Similarly, adding `-v` to the `macro_default` file causes the command:

```
mcc -W 'lib:libfoo' -T link:lib foo.m  
to behave as though the command were:
```

```
mcc -v -W 'lib:libfoo' -T link:lib foo.m
```

Invoke MATLAB Build Options

In this section...
“Specify Full Path Names to Build MATLAB Code” on page 4-4
“Using Bundles to Build MATLAB Code” on page 4-5

Specify Full Path Names to Build MATLAB Code

If you specify a full path name to a MATLAB file on the `mcc` command line, the compiler

- 1 Breaks the full name into the corresponding path name and file names (`<path>` and `<file>`).
- 2 Replaces the full path name in the argument list with “`-I <path> <file>`”.

Specifying Full Path Names

For example:

```
mcc -m /home/user/myfile.m
```

would be treated as

```
mcc -m -I /home/user myfile.m
```

In rare situations, this behavior can lead to a potential source of confusion. For example, suppose you have two different MATLAB files that are both named `myfile.m` and they reside in `/home/user/dir1` and `/home/user/dir2`. The command

```
mcc -m -I /home/user/dir1 /home/user/dir2/myfile.m
```

would be equivalent to

```
mcc -m -I /home/user/dir1 -I /home/user/dir2 myfile.m
```

The compiler finds the `myfile.m` in `dir1` and compiles it instead of the one in `dir2` because of the behavior of the `-I` option. If you are concerned that this might be happening, you can specify the `-v` option and then see which MATLAB file the compiler parses. The `-v` option prints the full path name to the MATLAB file during the dependency analysis phase.

Note The compiler produces a warning (`specified_file_mismatch`) if a file with a full path name is included on the command line and the compiler finds it somewhere else.

Using Bundles to Build MATLAB Code

Bundles provide a convenient way to group sets of compiler options and recall them as needed. The syntax of the bundle option is:

```
-B <bundle>[:<a1>,<a2>,...,<an>]
```

where `bundle` is either a predefined string such as `cpplib` or `csharedlib` or the name of a file that contains a set of `mcc` command-line options, arguments, filenames, and/or other `-B` options.

A bundle can include replacement parameters for compiler options that accept names and version numbers. For example, the bundle for C shared libraries, `csharedlib`, consists of:

```
-W lib:%1% -T link:lib
```

To invoke the compiler to produce the C shared library `mysharedlib` use:

```
mcc -B csharedlib:mysharedlib myfile.m myfile2.m
```

In general, each `%n%` in the bundle will be replaced with the corresponding option specified to the bundle. Use `%%` to include a `%` character. It is an error to pass too many or too few options to the bundle.

Note You can use the `-B` option with a replacement expression as is at the DOS or UNIX[®] prompt. If more than one parameter is passed, you must enclose the expression that follows the `-B` in single quotes. For example,

```
>>mcc -B csharedlib:libtimefun weekday data tic calendar toc
```

can be used as is at the MATLAB prompt because `libtimefun` is the only parameter being passed. If the example had two or more parameters, then the quotes would be necessary as in

```
>>mcc -B 'cexcel:component,class,1.0' ...
weekday data tic calendar toc
```

Available Bundle Files

Bundle File	Creates	Contents
cpplib	C++ library	-W cpplib: <i>library_name</i> -T link:lib
csharedlib	C library	-W lib: <i>library_name</i> -T link:lib
ccom	COM component	-W com: <i>component_name,className,version</i> -T link:lib
cexcel	Excel Add-in	-W excel: <i>addin_name,className,version</i> -T link:lib
cjava	Java package	-W java: <i>packageName,className</i>
dotnet	.NET assembly	-W dotnet: <i>assembly_name,className,framework_version,security</i> T link:lib

MATLAB Runtime Component Cache and Deployable Archive Embedding

In this section...

“Overriding Default Behavior” on page 4-8

“For More Information” on page 4-8

Deployable archive data is automatically embedded directly in compiled components and extracted to a temporary folder.

Automatic embedding enables usage of MATLAB Runtime Component Cache features through environment variables.

These variables allow you to specify the following:

- Define the default location where you want the deployable archive to be automatically extracted
- Add diagnostic error printing options that can be used when automatically extracting the deployable archive, for troubleshooting purposes
- Tuning the MATLAB Runtime component cache size for performance reasons.

Use the following environment variables to change these settings.

Environment Variable	Purpose	Notes
MCR_CACHE_ROOT	When set to the location of where you want the deployable archive to be extracted, this variable overrides the default per-user component cache location.	Does not apply
MCR_CACHE_VERBOSE	When set to any value, this variable prints logging details about the component cache for diagnostic reasons. This can be very helpful if problems are encountered during deployable archive extraction.	Logging details are turned off by default (for example, when this variable has no value).

Environment Variable	Purpose	Notes
MCR_CACHE_SIZE	When set, this variable overrides the default component cache size.	The initial limit for this variable is 32M (megabytes). This may, however, be changed after you have set the variable the first time. Edit the file <code>.max_size</code> , which resides in the file designated by running the <code>mcrcachedir</code> command, with the desired cache size limit.

You can override this automatic embedding and extraction behavior by compiling with the “Overriding Default Behavior” on page 4-8 option.

Caution If you run `mcc` specifying conflicting wrapper and target types, the deployable archive will not be embedded into the generated component. For example, if you run:

```
mcc -W lib:myLib -T link:exe test.m test.c
```

the generated `test.exe` will not have the deployable archive embedded in it, as if you had specified a `-C` option to the command line.

Overriding Default Behavior

To extract the deployable archive in a manner prior to R2008b, alongside the compiled .NET assembly, compile using the `mcc`'s `-C` option.

You might want to use this option to troubleshoot problems with the deployable archive, for example, as the log and diagnostic messages are much more visible.

For More Information

For more information about the deployable archive, see “Deployable Archive”.

Functions

productionServerCompiler

Test, build and package functions for use with MATLAB Production Server

Syntax

```
productionServerCompiler  
productionServerCompiler project_name  
productionServerCompiler -build project_name  
productionServerCompiler -package project_name
```

Description

`productionServerCompiler` opens the Production Server Compiler app for the creation of a new compiler project.

`productionServerCompiler project_name` opens the appropriate compiler app with the project preloaded.

`productionServerCompiler -build project_name` runs the appropriate compiler app to build the specified project. The installer is not generated.

`productionServerCompiler -package project_name` runs the appropriate compiler app to build and package the specified project. The installer is generated.

Examples

Create a New Production Server Project

Open the Production Server Compiler app to create a new project.

```
productionServerCompiler
```

Package a Deployable Archive using an Existing Project

Open the appropriate compiler app to package an existing project file.

```
productionServerCompiler -package my_magic
```

Input Arguments

project_name — name of the project to be compiled

string

Specify the name of a previously saved project. The project must be on the current path.

Introduced in R2014a

deploytool

Compile and package functions for external deployment

Syntax

```
deploytool  
deploytool project_name  
deploytool -build project_name  
deploytool -package project_name
```

Description

deploytool opens a list of the compiler apps.

deploytool project_name opens the appropriate compiler app with the project preloaded.

deploytool -build project_name runs the appropriate compiler app to build the specified project. The installer is not generated.

deploytool -package project_name runs the appropriate compiler app to build and package the specified project. The installer is generated.

Examples

Create a New Compiler Project

Open the compiler to create a new project.

```
deploytool
```

Package an Application using an Existing Project

Open the compiler to build a new application using an existing project.

```
deploytool -package my_magic
```

Input Arguments

project_name — name of the project to be compiled

string

Specify the name of a previously saved project. The project must be on the current path.

mcc

Compile MATLAB functions for deployment

Syntax

```

mcc -e | -m [-a filename...] [-B filename [:arg...]] [-C] [-d outFolder] [-f filename] [-g] [-I directory...] [-K] [-M string] [-N] [-o filename] [-p path...] [-R option] [-v] [-w option [:msg]] [-Y filename] mfilename
mcc -l [-a filename...] [-B filename [:arg...]] [-C] [-d outFolder] [-f filename] [-g] [-I directory...] [-K] [-M string] [-N] [-o filename] [-p path...] [-R option] [-v] [-w option [:msg]] [-Y filename] mfilename...
mcc -c [-a filename...] [-B filename [:arg...]] [-C] [-d outFolder] [-f filename] [-g] [-I directory...] [-K] [-M string] [-N] [-o filename] [-p path...] [-R option] [-v] [-w option [:msg]] [-Y filename] mfilename...
mcc -W cpplib:library_name -T link:lib [-a filename...] [-B filename [:arg...]] [-C] [-d outFolder] [-f filename] [-g] [-I directory...] [-K] [-M string] [-N] [-o filename] [-p path...] [-R option] [-S] [-v] [-w option [:msg]] [-Y filename] mfilename...
mcc -W dotnet:assembly_name, [className], [framework_version], security, remote_type -T link:lib [-a filename...] [-B filename [:arg...]] [-C] [-d outFolder] [-f filename] [-I directory...] [-K] [-M string] [-N] [-p path...] [-R option] [-S] [-v] [-w option [:msg]] [-Y filename] mfilename... [class{className: [mfilename...]}...]
mcc -W excel:addin_name, [className], [version] -T link:lib [-a filename...] [-b] [-B filename [:arg...]] [-C] [-d outFolder] [-f filename] [-I directory...] [-K] [-M string] [-N] [-p path...] [-R option] [-u] [-v] [-w option [:msg]] [-Y filename] mfilename...
mcc -W 'java:packageName, [className]' [-a filename...] [-b] [-B filename [:arg...]] [-C] [-d outFolder] [-f filename] [-I directory...] [-K] [-M string] [-N] [-p path...] [-R option] [-S] [-v] [-w option [:msg]] [-Y filename] mfilename... [class{className: [mfilename1,mfilename2,...]}...]
mcc -W CTF:archive_name [-a filename...] [-b] [-B filename [:arg...]] [-d outFolder] [-f filename] [-I directory...] [-K] [-M string] [-N] [-p path...] [-R option] [-S] [-v] [-w option [:msg]] [-Y filename] mfilename... [class{className: [mfilename1,mfilename2,...]}...]
mcc -W mpsxl:archive_name, [className], [version] -T link:lib [-replaceBlankWithNaN] [-convertDateToString] [-replaceNaNWithZero] [-convertNumericToDate] [-a filename...] [-b] [-B filename [:arg...]] [-d outFolder] [-f filename] [-I directory...] [-K] [-M string] [-N] [-p path...] [-R option]

```

```

[-S] [-v] [-w option [:msg]] [-Y filename] filename... [class{className:
[mfilename1,mfilename2,...]}...]
mcc -H -W mpsxl:archive_name, [className], [version] -T link:lib [-
replaceBlankWithNaN] [-convertDateToString] [-replaceNaNWithZero] [-
convertNumericToDate] [-a filename...] [-b] [-B filename [:arg...]] [-d outFolder] [-f
filename] [-I directory...] [-K] [-M string] [-N] [-p path...] [-R option] [-S] [-v] [-w
option [:msg]] [-Y filename] filename... [class{className: [mfilename...]}...]
mcc -H -W hadoop:archive_name,CONFIG:configFile [-a filename...] [-b] [-B
filename [:arg...]] [-d outFolder] [-f filename] [-I directory...] [-K] [-M string]
[-N] [-p path...] [-R option] [-S] [-v] [-w option [:msg]] [-Y filename] filename...
[class{className: [mfilename1,mfilename2,...]}...]
mcc -?

```

Description

`mcc -m mfilename` compiles the function into a standalone application.

This is equivalent to `-W main -T link:exe`.

`mcc -e mfilename` compiles the function into a standalone application that does not open an MS-DOS[®] command window.

This is equivalent to `-W WinMain -T link:exe`.

`mcc -l mfilename...` compiles the listed functions into a C shared library and generates C wrapper code for integration with other applications.

This is equivalent to `-W lib:libname -T link:lib`.

`mcc -c mfilename...` generates C wrapper code for the listed functions.

This is equivalent to `-W lib:libname -T codegen`.

`mcc -W cpplib:library_name -T link:lib mfilename...` compiles the listed functions into a C++ shared library and generates C++ wrapper code for integration with other applications.

`mcc -W dotnet:assembly_name,className,framework_version,security,remote_type -T link:lib mfilename...` creates a .NET assembly from the specified files.

- *assembly_name* — Specifies the name of the assembly and its namespace, which is a period-separated list, such as `companyname.groupname.component`.
- *className* — Specifies the name of the .NET class to be created.
- *framework_version* — Specifies the version of the Microsoft .NET Framework you want to use to compile the assembly. Specify either:
 - `0.0` — Use the latest supported version on the target machine.
 - *version_major.version_minor* — Use a specific version of the framework.

Features are often version-specific. Consult the documentation for the feature you are implementing to get the Microsoft .NET Framework version requirements.

- *security* — Specifies whether the assembly to be created is a private assembly or a shared assembly.
 - To create a private assembly, specify `Private`.
 - To create a shared assembly, specify the full path to the encryption key file used to sign the assembly.
- *remote_type* — Specifies the remoting type of the assembly. Values are `remote` and `local`.

By default, the compiler generates a single class with a method for each function specified on the command line. You can instruct the compiler to create multiple classes using `class{className:mfilename...}...className` specifies the name of the class to create using *mfilename*.

`mcc -W excel:addin_name,className,version -T link:lib mfilename...`
creates a Microsoft Excel add-in from the specified files.

- *addin_name* — Specifies the name of the addin and its namespace, which is a period-separated list, such as `companyname.groupname.component`.
- *className* — Specifies the name of the class to be created. If you do not specify the class name, `mcc` uses the *addin_name* as the default.
- *version* — Specifies the version of the add-in specified as *major.minor*.
 - *major* — Specifies the major version number. If you do not specify a version number, `mcc` uses the latest version.
 - *minor* — Specifies the minor version number. If you do not specify a version number, `mcc` uses the latest version.

`mcc -W 'java:packageName,className' mfilename...` creates a Java package from the specified files.

- *packageName* — Specifies the name of the Java package and its namespace, which is a period-separated list, such as `companyname.groupname.component`.
- *className* — Specifies the name of the class to be created. If you do not specify the class name, `mcc` uses the last item in *packageName*.

By default, the compiler generates a single class with a method for each function specified on the command line. You can instruct the compiler to create multiple classes using `class{className:mfilename...}...className` specifies the name of the class to create using *mfilename*.

`mcc -W CTF:archive_name` instructs the compiler to create a deployable archive that is deployable in a MATLAB Production Server instance.

`mcc -W mpsxl:addin_name,className,version input_marshaling_options output_marshaling_options -T link:lib mfilename...` creates an MATLAB Production Server integrated Microsoft Excel add-in from the specified files.

- *addin_name* — Specifies the name of the add-in and its namespace, which is a period-separated list, such as `companyname.groupname.component`.
- *className* — Specifies the name of the class to be created. If you do not specify the class name, `mcc` uses the *addin_name* as the default.
- *version* — Specifies the version of the add-in specified as *major.minor*.
 - *major* — Specifies the major version number. If you do not specify a version number, `mcc` uses the latest version.
 - *minor* — Specifies the minor version number. If you do not specify a version number, `mcc` uses the latest version.
- *input_marshaling_flags* — Specifies options for how data is marshaled between Microsoft Excel and MATLAB.
 - `-replaceBlankWithNaN` — Specifies that a blank in Microsoft Excel is marshaled into NaN in MATLAB. If you do not specify this flag, blanks are marshaled into 0.
 - `-convertDateToString` — Specifies that dates in Microsoft Excel are marshaled into MATLAB strings. If you do not specify this flag, dates are marshaled into MATLAB doubles.
- *output_marshaling_flags* — Specifies options for how data is marshaled between MATLAB and Microsoft Excel.

- `-replaceNaNWithZero` — Specifies that NaN in MATLAB is marshaled into a 0 in Microsoft Excel. If you do not specify this flag, NaN is marshaled into #QNAN in Visual Basic®.
- `-convertNumericToDate` — Specifies that MATLAB numeric values are marshaled into Microsoft Excel dates. If you do not specify this flag, Microsoft Excel does not receive dates as output.

`mcc -H -W hadoop:archiveName,CONFIG:configFile` generates deployable archive that can be run as a job by Hadoop®.

- *archiveName* — Specifies the name of the generated archive.
- *configFile* — Specifies the path to the Hadoop settings file. See “Hadoop Settings File”.

`mcc -?` displays help.

Tip You can issue the `mcc` command either from the MATLAB command prompt or the DOS or UNIX command line.

Options

-a Add to Archive

Add files to the deployable archive using

`-a path`

to specify files to be added to the deployable archive. Multiple `-a` options are permitted.

If a file name is specified with `-a`, the compiler looks for these files on the MATLAB path, so specifying the full path name is optional. These files are not passed to `mbuild`, so you can include files such as data files.

If a folder name is specified with the `-a` option, the entire contents of that folder are added recursively to the deployable archive. For example

```
mcc -m hello.m -a ./testdir
```

specifies that all files in `testdir`, as well as all files in subfolders of `testdir`, are added to the deployable archive, and the folder subtree in `testdir` is preserved in the deployable archive.

If a wildcard pattern is included in the file name, only the files in the folder that match the pattern are added to the deployable archive and subfolders of the given path are not processed recursively. For example

```
mcc -m hello.m -a ./testdir/*
```

specifies that all files in `./testdir` are added to the deployable archive and subfolders under `./testdir` are not processed recursively.

```
mcc -m hello.m -a ./testdir/*.m
```

specifies that all files with the extension `.m` under `./testdir` are added to the deployable archive and subfolders of `./testdir` are not processed recursively.

Note: Currently, `*` is the only supported wildcard.

All files added to the deployable archive using `-a` that do not appear on the MATLAB path at the time of compilation causes a path entry to be added to the deployed application's run-time path so that they appear on the path when the deployed code executes.

When files are included, the absolute path for the DLL and header files is changed. The files are placed in the `.\exe_mcr\` folder when the deployable archive is expanded. The file is not placed in the local folder. This folder is created from the deployable archive the first time the application is executed. The `isdeployed` function is provided to help you accommodate this difference in deployed mode.

The `-a` switch also creates a `.auth` file for authorization purposes. It ensures that the executable looks for the DLL- and H-files in the `exe_mcr\exe` folder.

Caution If you use the `-a` flag to include a file that is not on the MATLAB path, the folder containing the file is added to the MATLAB dependency analysis path. As a result, other files from that folder might be included in the compiled application.

Note: If the `-a` flag is used to include custom Java classes, standalone applications work without any need to change the `classpath` as long as the Java class is not a member of a package. The same applies for JAR files. However, if the class being added is a member of a package, the MATLAB code needs to make an appropriate call to `javaaddpath` to update the `classpath` with the parent folder of the package.

-b Generate Excel Compatible Formula Function

Generate a Visual Basic file (`.bas`) containing the Microsoft Excel Formula Function interface to the COM object generated by MATLAB Compiler. When imported into the workbook Visual Basic code, this code allows the MATLAB function to be seen as a cell formula function.

-B Specify Bundle

Replace the file on the `mcc` command line with the contents of the specified file. Use

```
-B filename[:<a1>,<a2>,...,<an>]
```

The bundle `filename` should contain only `mcc` command-line options and corresponding arguments and/or other file names. The file might contain other `-B` options. A bundle can include replacement parameters for compiler options that accept names and version numbers. See “Using Bundles to Build MATLAB Code”.

-C Do Not Embed Deployable Archive

Do not embed the deployable archive in binaries.

-d Specified Folder for Output

Place output in a specified folder. Use

```
-d outFolder
```

to direct the generated files to *outFolder*.

-f Specified Options File

Override the default options file with the specified options file. Use

`-f filename`

to specify `filename` as the options file when calling `mbuild`. This option lets you use different ANSI compilers for different invocations of the compiler. This option is a direct pass-through to `mbuild`.

-g Generate Debugging Information

Include debugging symbol information for the C/C++ code generated by MATLAB Compiler SDK™. It also causes `mbuild` to pass appropriate debugging flags to the system C/C++ compiler. The debug option lets you backtrace up to the point where you can identify if the failure occurred in the initialization of MATLAB Runtime, the function call, or the termination routine. This option does not let you debug your MATLAB files with a C/C++ debugger.

-G Debug Only

Same as `-g`.

-I Add Folder to Include Path

Add a new folder path to the list of included folders. Each `-I` option adds a folder to the beginning of the list of paths to search. For example,

```
-I <directory1> -I <directory2>
```

sets up the search path so that `directory1` is searched first for MATLAB files, followed by `directory2`. This option is important for standalone compilation where the MATLAB path is not available.

If used in conjunction with the `-N` option, the `-I` option will add the folder to the compilation path in the same position where it appeared in the MATLAB path rather than at the head of the path.

-K Preserve Partial Output Files

Direct `mcc` not to delete output files if the compilation ends prematurely, due to error.

The default behavior of `mcc` is to dispose of any partial output if the command fails to execute successfully.

-M Direct Pass Through

Define compile-time options. Use

```
-M string
```

to pass `string` directly to `mbuild`. This provides a useful mechanism for defining compile-time options, e.g., `-M "-Dmacro=value"`.

Note: Multiple `-M` options do not accumulate; only the rightmost `-M` option is used.

-N Clear Path

Passing `-N` effectively clears the path of all folders except the following core folders (this list is subject to change over time):

- `matlabroot\toolbox\matlab`
- `matlabroot\toolbox\local`
- `matlabroot\toolbox\compiler`

It also retains all subfolders of the above list that appear on the MATLAB path at compile time. Including `-N` on the command line lets you replace folders from the original path, while retaining the relative ordering of the included folders. All subfolders of the included folders that appear on the original path are also included. In addition, the `-N` option retains all folders that you included on the path that are not under `matlabroot\toolbox`.

When using the `-N` option, use the `-I` option to force inclusion of a folder, which will be placed at the head of the compilation path. Use the `-p` option to conditionally include folders and their subfolders; if they are present in the MATLAB path, they will appear in the compilation path, in the same order.

-o Specify Output Name

Specify the name of the final executable (standalone applications only). Use

```
-o outputfile
```

to name the final executable output of MATLAB Compiler. A suitable, possibly platform-dependent, extension is added to the specified name (e.g., `.exe` for Windows standalone applications).

-p Add Folder to Path

Use in conjunction with the required option `-N` to add specific folders (and subfolders) under `matlabroot\toolbox` to the compilation MATLAB path in the same order in which they appear in the MATLAB path. Use the syntax

`-N -p directory`

where `directory` is the folder to be included. If `directory` is not an absolute path, it is assumed to be under the current working folder.

- If a folder is included with `-p` that is on the original MATLAB path, the folder and all its subfolders that appear on the original path are added to the compilation path in the same order.
- If a folder is included with `-p` that is not on the original MATLAB path, that folder is ignored. (You can use `-I` to force its inclusion.)

-R Run-Time

Provides MATLAB Runtime options. The syntax is as follows:

`-R option`

Option	Description	Target
<code>-logfile,</code>	Specify a log file name.	MATLAB Compiler MATLAB Compiler SDK
<code>-nodisplay</code>	Suppress the MATLAB <code>nodisplay</code> run-time warning.	MATLAB Compiler MATLAB Compiler SDK
<code>-nojvm</code>	Do not use the Java Virtual Machine (JVM).	MATLAB Compiler MATLAB Compiler SDK
<code>-startmsg</code>	Customizable user message displayed at initialization time.	MATLAB Compiler Standalone Applications

Option	Description	Target
-complete	Customizable user message displayed when initialization is complete.	MATLAB Compiler Standalone Applications

Caution When running on Mac OS X, if `-nodisplay` is used as one of the options included in `mclInitializeApplication`, then the call to `mclInitializeApplication` must occur before calling `mclRunMain`.

-S Create Singleton MATLAB Runtime Context

The standard behavior for the MATLAB Runtime is that every instance of a class gets its own runtime context. This runtime context includes a global MATLAB workspace for variables such as the path and a base workspace for each function in the class. If multiple instances of a class are created, each instance gets an independent context. This ensures that changes made to the global, or base, workspace in one instance of the class does not affect other instances of the same class.

In a singleton MATLAB Runtime, all instances of a class share the runtime context. If multiple instances of a class are created, they use the runtime context created by the first instance. This saves start up time and some resources. However, any changes made to the global workspace or the base workspace by one instance impacts all of the class instances. For example, if `instance1` creates a global variable `A` in a singleton MATLAB Runtime, the `instance2` will be able to use variable `A`.

Singleton MATLAB Runtime is only supported by the following products on these specific targets:

Target supported by Singleton MATLAB Runtime	Create a Singleton MATLAB Runtime by...
Excel add-in	Default behavior for target is singleton runtime. You do not need to perform other steps.
.NET assembly	Default behavior for target is singleton runtime. You do not need to perform other steps.
COM component	<ul style="list-style-type: none"> Using the Library Compiler app, click Settings and add -S to the Additional flags to pass to mcc field. Using <code>mcc</code> pass the <code>-S</code> flag.
Java packages	

-T Specify Target Stage

Specify the output target phase and type.

Use the syntax `-T target` to define the output type. Target values are as follow.

Target	Description
<code>compile:exe</code>	Generate a C/C++ wrapper file and compile C/C++ files to object form suitable for linking into a standalone application.
<code>compile:lib</code>	Generate a C/C++ wrapper file and compile C/C++ files to object form suitable for linking into a shared library/DLL.
<code>link:exe</code>	Same as <code>compile:exe</code> and links object files into a standalone application.
<code>link:lib</code>	Same as <code>compile:lib</code> and links object files into a shared library/DLL.

-u Register COM Component for the Current User

Register COM component for the current user only on the development machine. The argument applies only for generic COM component and Microsoft Excel add-in targets only.

-v Verbose

Display the compilation steps, including:

- MATLAB Compiler version number
- The source file names as they are processed
- The names of the generated output files as they are created
- The invocation of `mbuild`

The `-v` option passes the `-v` option to `mbuild` and displays information about `mbuild`.

-w Warning Messages

Display warning messages. Use the syntax

-w option [:<msg>]

to control the display of warnings. This table lists the syntaxes.

Syntax	Description
-w list	List all of the possible warning that mcc can generate.
-w enable	Enable complete warnings.
-w disable[:<string>]	Disable specific warnings associated with <string>. See “Warning Messages” for a list of the <string> values. Omit the optional <string> to apply the disable action to all warnings.
-w enable[:<string>]	Enable specific warnings associated with <string>. See “Warning Messages” for a list of the <string> values. Omit the optional <string> to apply the enable action to all warnings.
-w error[:<string>]	Treat specific warnings associated with <string> as an error. Omit the optional <string> to apply the error action to all warnings.
-w off[:<string>] [<filename>]	Turn warnings off for specific error messages defined by <string>. You can also narrow scope by specifying warnings be turned off when generated by specific <filename>s.
-w on[:<string>] [<filename>]	Turn warnings on for specific error messages defined by <string>. You can also narrow scope by specifying warnings be turned on when generated by specific <filename>s.

It is also possible to turn warnings on or off in your MATLAB code.

For example, to turn warnings off for deployed applications (specified using `isdeployed`) in your `startup.m`, you write:

```
if isdeployed
    warning off
end
```

To turn warnings on for deployed applications, you write:

```
if isdeployed
```

```
    warning on  
end
```

-Y License File

Use

```
-Y license.lic
```

to override the default license file with the specified argument.

See Also

`mbuild`

Apps — Alphabetical List

Production Server Compiler

Test and Package MATLAB functions for deployment to MATLAB Production Server

Description

The **Production Server Compiler app** tests the integration of client code with MATLAB functions. It also packages MATLAB functions into archives for deployment to MATLAB Production Server.

Open the Production Server Compiler App

- MATLAB Toolstrip: On the **Apps** tab, under **Application Deployment**, click the app icon.
- MATLAB command prompt: Enter `productionServerCompiler`.

Examples

- “Create a Deployable Archive for MATLAB Production Server”
- “Build Excel Add-In and Deployable Archive”

More About

- “Deployable Archive Creation”
- “Compile MATLAB Functions”

Parameters

Type

Specify the type of application the packaged code will be used in.

Default: Deployable Archive (.ctf)

Settings

Deployable Archive (.ctf)

Packaged code is being used as part of an application where the client-side portion of the application is written using one of the MATLAB Production Server APIs.

Deployable Archive with Excel Integration

Packaged code is being used as part of an application where the client-side portion of the application is a generated Excel add-in. The app generates the Excel add-in as well.

Exported Functions

Specify the MATLAB functions to package.

Settings

Specify the output folders for the packaged code.

Default:

- Testing folder: `for_testing`
- Redistribution folder: `for_redistribution`

Archive Information

Specify the a name for the packaged application.

Default: Name of the first file in the **Exported Functions** list.

Files required for your archive to run

Specify the MATLAB files and data files that the application requires to run. The listed files are packaged into the generated archive.

Default: The list of files generated by the built-in dependency analysis tool.

Files installed with your application

Specify additional files that are installed by the generated installer. The listed files are installed in the same folder as the installed archive.

Default: The generated archive (.ctf) file and a readme.txt file.

Programmatic Use

productionServerCompiler

Introduced in R2013b