MATLAB® Production Server™

Code Deployment

MATLAB®

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

Revision History

Contents

Customizing a Compiler Project

3

Advanced Uses of the Command Line Compiler

4

Functions

5

Apps — Alphabetical List

6

Persistence

7

Persistence Functions

8

Write Deployable MATLAB Code

MATLAB Coding Guidelines

When writing MATLAB code for deployment to MATLAB Production Server you must adhere to the same to the same guidelines as when writing code for deployment with MATLAB Compiler™ or MATLAB Compiler SDK™. In addition, code deployed to MATLAB Production Server must adhere to additional guidelines:

- functions cannot depend on nor change MATLAB state.

  Functions deployed with MATLAB Production Server may not always execute on the same instance of the MATLAB Runtime. Each worker access a different MATLAB Runtime instance.
- explicitly use `varargin` and `varargout` for functions with variable inputs and outputs.
- avoid MATLAB figure or GUI code.

  Deployed MATLAB code runs on the server, any figures or GUIs created during runtime will show up on the server machine, not the client machine. If figures or GUIs are required to run to create the function results, make sure to close these figures at the end of your code to avoid left over windows and leaking resources on the server.

See Also

More About

- "State-Dependent Functions" on page 1-3
- "Write Deployable MATLAB Code" (MATLAB Compiler)

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

2

Create a Deployable Archive from MATLAB Code

Create a Deployable Archive for MATLAB Production Server

Supported platform: Windows®, Linux®, Mac

This example shows how to create a deployable archive from a MATLAB function. You can then give the generated archive to a system administrator to deploy it on the MATLAB Production Server environment.

Create a Function In MATLAB

In MATLAB, examine the MATLAB program that you want to package.

For this example, write a function `addmatrix.m` as follows.

```
function a = addmatrix(a1, a2)
%ADDMATRIX Add two matrices
%    This function adds the two matrices passed as input. This function is
%    used to demonstrate the functionality of MATLAB Compiler SDK. Refer to the
%    documentation for MATLAB Compiler SDK C/C++ shared library for more information.

% Copyright 2003-2017 The MathWorks, Inc.

a = a1 + a2;
```

At the MATLAB command prompt, enter `addmatrix([1 4 7; 2 5 8; 3 6 9], [1 4 7; 2 5 8; 3 6 9])`.

The output is:

```
 ans =
     2     8    14
     4    10    16
     6    12    18
```

Create a Deployable Archive with Production Server Compiler App

1 On the **MATLAB Apps** tab, on the far right of the **Apps** section, click the arrow. In **Application Deployment**, click **Production Server Compiler**. In the **MATLAB Compiler SDK** project window, click **Deployable Archive (.ctf)**.
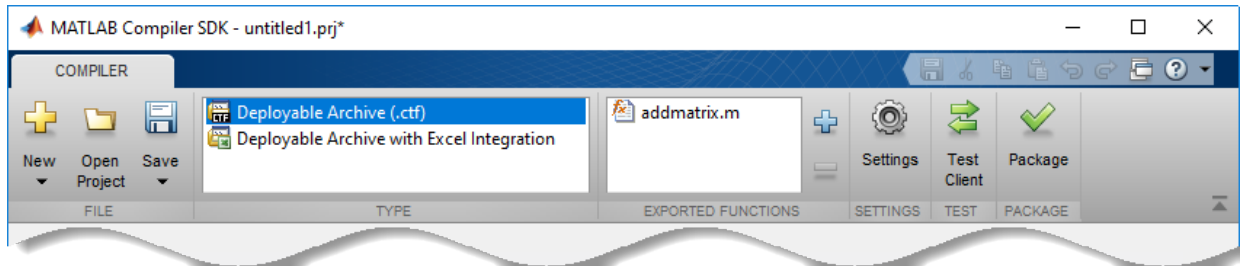
Alternatively, you can open the **Production Server Compiler** app by entering `productionServerCompiler` at the MATLAB prompt.

2   In the **MATLAB Compiler SDK** project window, specify the main file of the MATLAB application that you want to deploy.

    1   In the **Exported Functions** section, click .

    2   In the **Add Files** window, browse to the example folder, and select the function you want to package.

       Click **Open**.

    The function `addmatrix.m` is added to the list of main files.
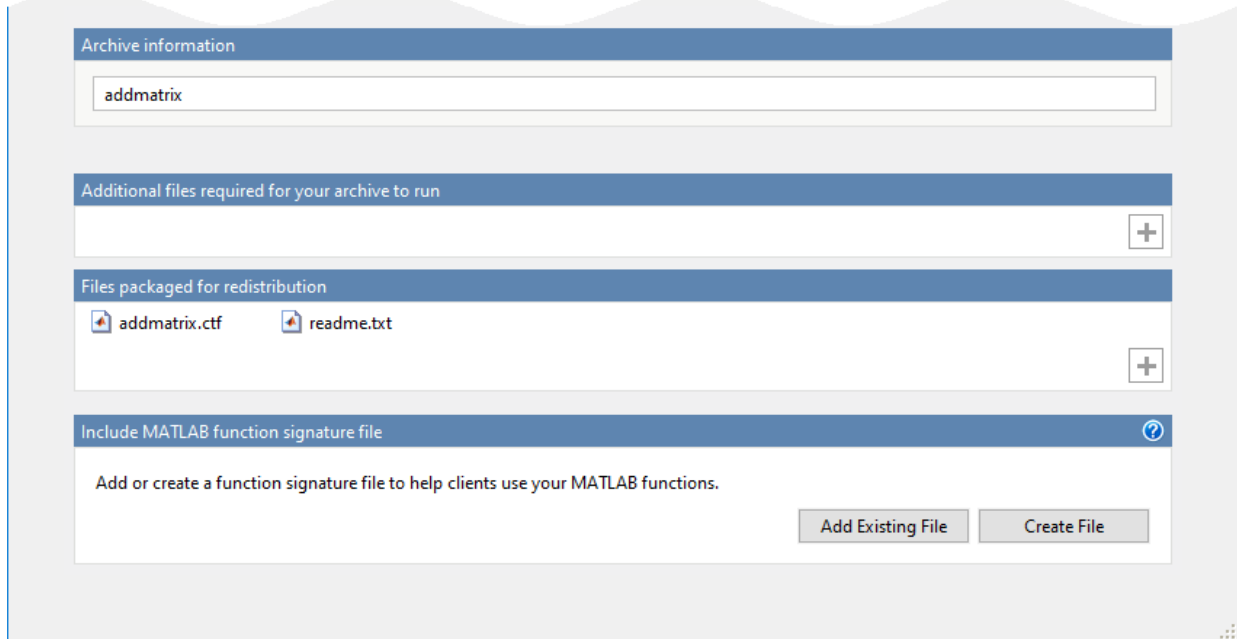
Customize the Application and Its Appearance

You can customize your deployable archive, and add more information about the application as follows:

- **Archive information** — Editable information about the deployed archive.

- **Additional files required for your archive to run** — Additional files required to run the generated archive. These files are included in the generated archive installer. See "Manage Required Files in Compiler Project" (MATLAB Compiler SDK).

- **Files packaged for redistribution** — Files that are installed with your archive. These files include:

  - Generated deployable archive

  - Generated `readme.txt`

See "Specify Files to Install with Application" (MATLAB Compiler SDK).

- **Include MATLAB function signature file** — Add or create a function signature file to help clients use your MATLAB functions. See "MATLAB Function Signatures in JSON".

Package the Application

1 To generate the packaged application, click **Package**.

In the Save Project dialog box, specify the location to save the project.
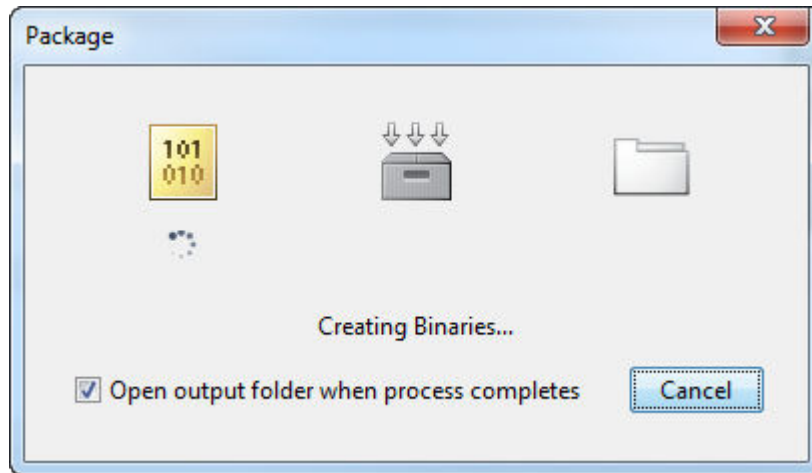
2   In the **Package** dialog box, verify that **Open output folder when process completes** is selected.

When the deployment process is complete, examine the generated output.

- `for_redistribution` — Folder containing the archive *archiveName*.ctf
- `for_testing` — Folder containing the raw generated files to create the installer
- `PackagingLog.txt` — Log file generated by MATLAB Compiler

See Also
deploytool | mcc | productionServerCompiler

More About
- Production Server Compiler
- "MATLAB Function Signatures in JSON"

Package Deployable Archives from Command Line

In this section...
"Execute Compiler Projects with deploytool" on page 2-6
"Package a Deployable Archive with mcc" on page 2-6
"Differences Between Compiler Apps and Command Line" on page 2-7

You can package deployable archives at the MATLAB prompt or your system prompt using either of these commands.

- `deploytool` invokes the Application Compiler app to execute a saved compiler project.
- `mcc` invokes the MATLAB Compiler to create a deployable application at the command prompt.

Execute Compiler Projects with deploytool

The `deploytool` command has two flags that invoke one of the compiler apps to package an already existing project without opening a window.

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

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

Package a Deployable Archive with mcc

The `mcc` command invokes the MATLAB Compiler and provides fine-level control over the packaging 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 packaging deployable archives, you can also use the following options.

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

Differences Between Compiler Apps and Command Line

You perform the same functions using either the compiler apps or the mcc command-line interface. The interactive menus and dialog boxes used in the compiler apps build mcc commands that are customized to your specification. As such, your MATLAB code is processed the same way as if you were packaging it using mcc.

If you know the commands for the type of application you want to deploy and do not require an installer, it is faster to execute mcc than go through the compiler app workflow.

Compiler app advantages include:

- You can perform related deployment tasks with a single intuitive interface.
- You can maintain related information in a convenient project file.
- Your project state persists between sessions.
- You can load previously stored compiler projects from a prepopulated menu.
- You can package applications for distribution.

See Also

deploytool | mcc

More About

- "Package Deployable Archives with Production Server Compiler App" (MATLAB Compiler SDK)

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).

3

Customizing a Compiler Project

Customize the Application

You can customize the application in several ways: customize the installer, manage files in the project, or add a custom installer path using the **Application Compiler** app or the **Library Compiler** app.

Customize the Installer

Change Application Icon

To change the default icon, click the graphic to the left of the **Library name** or **Application name** field to preview the icon.

Click **Select icon**, and locate the graphic file to use as the application icon. Select the **Use mask** option to fill any blank spaces around the icon with white or the **Use border** option to add a border around the icon.

To return to the main window, click **Save and Use**.

Add Library or Application Information

You can provide further information about your application as follows:

- Library/Application Name: The name of the installed MATLAB artifacts. For example, if the name is `foo`, the installed executable is `foo.exe`, and the Windows start menu entry is **foo**. The folder created for the application is *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: Name of the developer.

- Support email address: Email addess to use for contact information.

- Company name: 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: Brief summary describing the application.

- Description: Detailed explanation about the application.

All information is optional and, unless otherwise stated, is only displayed 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 splash screen displays after the installer has 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**. When the file explorer opens, locate and select a new image.

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

Change the Installation Path

This table lists the default path the installer uses when installing the packaged binaries onto a target system.

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**.
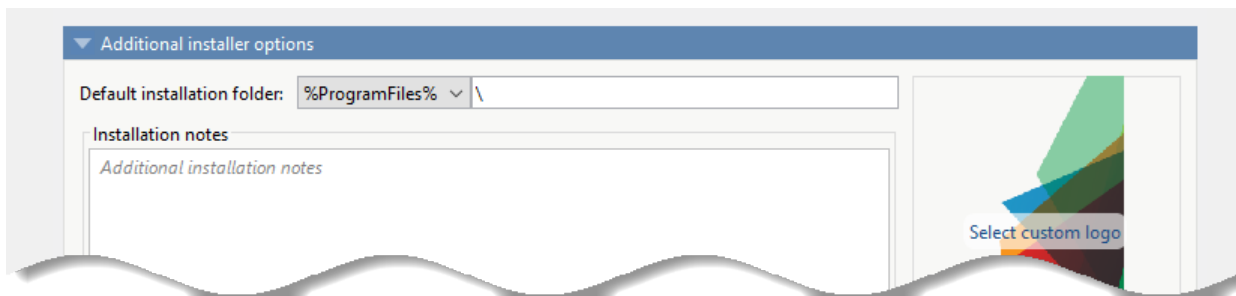
A text field specifying the path appended to the root folder is your installation folder. You can pick the root folder for the application installation folder. This table lists the optional custom root folders for each platform:

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

Change the Logo

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

You change the default image in **Additional Installer Options** by clicking **Select custom logo**. When the file explorer opens, locate and select a new image. 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 provide useful information concerning any additional setup that is required to use the installed binaries and instructions for how to run the application.

Manage Required Files in Compiler Project

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

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

To add files, click the plus button in the field, and select the file from the file explorer. To remove files, select the files, and press the **Delete** key.

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

Using mcc

If you are using `mcc` to package your MATLAB code, the compiler does not display a list of required files before running. Instead, it packages all 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 the files in `foo` and its subfolders to the list of required files.

Specify Files to Install with Application

The compiler packages 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 for your end user** section of the app.

To add files to the list, click [+], and select the file from the file explorer.

JAR files are added to the application class path as if you had called `javaaddpath`.

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 for your end user** are saved in the `application` folder.

Additional Runtime Settings

See Also

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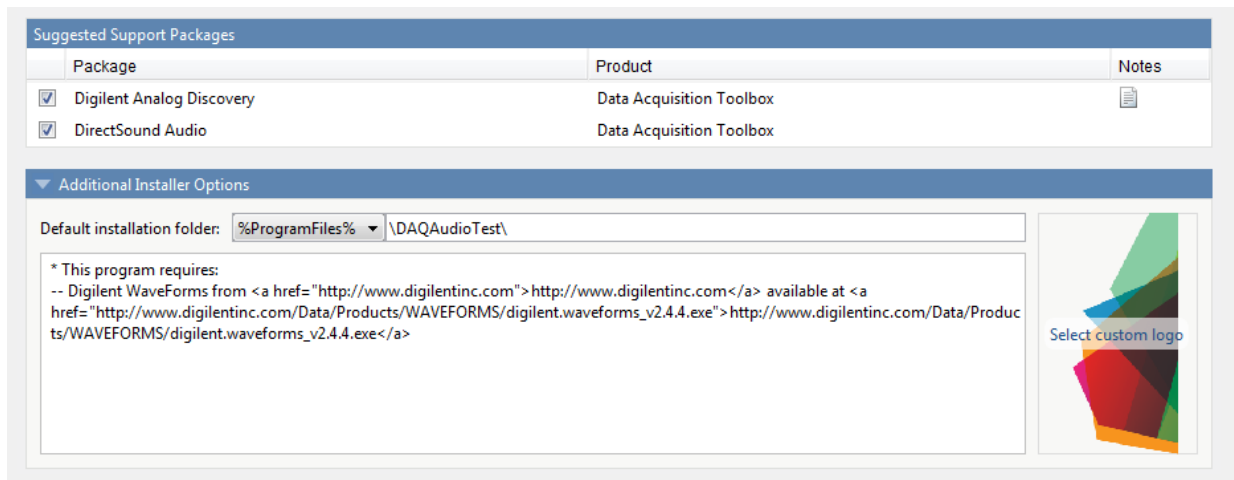
The list displays all installed support packages that your MATLAB code requires. The list is determined using these criteria:

- 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, use the -a flag with mcc command when packaging your MATLAB code to specify supporting files in the support package folder. For example, if your function uses the OS Generic Video Interface support package, run the following command:

```
mcc -m -v test.m -a C:\MATLAB\SupportPackages\R2016b\toolbox\daq\supportpackages\daqaud
```

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:*library_name* -T link:lib
csharedlib	C library	-W lib:*library_name* -T link:lib
ccom	COM component	-W com:*component_name*,*className*,*version* -T link:lib
cexcel	Excel Add-in	-W excel:*addin_name*,*className*,*version* -T link:lib
cjava	Java package	-W java:*packageName*,*className*
dotnet	.NET assembly	-W dotnet:*assembly_name*,*className*,*framework_version*,*security*,*remote_type* -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 4-9

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. This is true for embedded `.ctf` files only.	Does not apply

Environment Variable	Purpose	Notes
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).
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 `.max_size`, which resides in the file designated by running the `mcrcachedir` 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" (MATLAB Compiler).

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
character array or 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
character array or string

Name of the project to be compiled, specified as a character array or string.The project must be on the current path.

mcc

Compile MATLAB functions for deployment

Syntax

```
mcc options mfilename1 mfilename2...mfilenameN
```

```
mcc -W CTF:archive_name -U options mfilename1
mfilename2...mfilenameN
```

```
mcc -W mpsxl:addin_name,className,version input_marshaling_flags
output_marshaling_flags -T link:lib options mfilename1
mfilename2...mfilenameN
```

Description

`mcc options mfilename1 mfilename2...mfilenameN` compiles the functions as specified by the options.

The options used depend on the intended results of the compilation. For information on compiling:

- standalone applications, Excel add-ins, or Hadoop® jobs see `mcc` for MATLAB Compiler
- C/C++ shared libraries, .NET assemblies, Java packages, or Python® packages see `mcc` for MATLAB Compiler SDK

`mcc -W CTF:archive_name -U options mfilename1 mfilename2...mfilenameN` instructs the compiler to create a deployable archive (`.ctf` file) for use with a MATLAB Production Server instance.

The syntax also creates the server-side deployable archive (`.ctf` file) for Microsoft Excel add-ins.

`mcc -W mpsxl:addin_name,className,version input_marshaling_flags output_marshaling_flags -T link:lib options mfilename1 mfilename2...mfilenameN` creates a client-side Microsoft Excel add-in from the

specified files that can be used to send requests to MATLAB Production Server from Excel. Creating the client-side add-in *must* be preceded by creating a server-side deployable archive (`.ctf` file) from the specified files. A purely client side add-in is not viable.

- *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 mashaled 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 character vectors. 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 marshalled 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.

Examples

Input Arguments

mfilename1 mfilename2...mfilenameN — Files to be compiled
list of filenames

One or more files to be compiled, specified as a space-separated list of filenames.

options — Options for customizing the output
-a | -b | -B | -C | -d | -f | -g | -G | -I | -K | -m | -M | -n | -N | -o | -p | -R | -S | -T | -u | -U | -v | -w | -W | -Y

Options for customizing the output, specified as a list of character vectors or string scalars.

- **-a**

  Add files to the deployable archive using `-a path` to specify the files to be added. 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 its subfolders, are added to the deployable archive. The folder subtree in `testdir` is preserved in the deployable archive.

  If the filename includes a wildcard pattern, 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 `*` is the only supported wildcard.

When you add files to the archive using `-a` that do not appear on the MATLAB path at the time of compilation, a path entry is added to the application's run-time path so that they appear on the path when the deployed code executes.

When you use the `-a` option to specify a full path to a resource, the basic path is preserved, with some modifications, but relative to a subdirectory of the runtime cache directory, not to the user's local folder. The cache directory is created from the deployable archive the first time the application is executed. You can use the `isdeployed` function to determine whether the application is being run in deployed mode, and adjust the path accordingly. The `-a` option also creates a `.auth` file for authorization purposes.

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 you use the `-a` flag 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 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**

  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" (MATLAB Compiler SDK).

- **-C**

  Do not embed the deployable archive in binaries.

  Note The `-C` flag is ignored for Java libraries.

- **-d**

  Place output in a specified folder. Use

  ```
  -d outFolder
  ```

  to direct the generated files to *outFolder*.

- **-f**

  Override the default options file with the specified options file. It specifically applies to the C/C++ shared libraries, COM, and Excel targets. 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, -G**

  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.

- **-I**

  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 adds 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**

  Direct `mcc` to not 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 `mcc` to generate a standalone application.

- **-M**

  Define compile-time options. Use

  ```
  -M string
  ```

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

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

- **-n**

  The -n option automatically identifies numeric command line inputs and treats them as MATLAB doubles.

- **-N**

  Passing -N 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
- *matlabroot*\toolbox\shared\bigdata

Passing -N also retains all subfolders in this 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 is 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 appear in the compilation path in the same order.

- **-o**

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

  ```
  -o outputfile
  ```

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

- **-p**

  Use in conjunction with the option -N to add specific folders and subfolders under *matlabroot*\toolbox to the compilation MATLAB path. The files are added 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**

  Provide MATLAB Runtime options. This option is relevant only when building standalone applications using MATLAB Compiler. The syntax is as follows:

  -R *option*

Option	Description	Target
-logfile,*filename*	Specify a log file name.	MATLAB Compiler
-nodisplay	Suppress the MATLAB `nodisplay` run-time warning.	MATLAB Compiler
-nojvm	Do not use the Java Virtual Machine (JVM).	MATLAB Compiler
-startmsg	Customizable user message displayed at initialization time.	MATLAB Compiler Standalone Applications
-completemsg	Customizable user message displayed when initialization is complete.	MATLAB Compiler Standalone Applications

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

  Note If you specify the `-R` option for libraries created from MATLAB Compiler SDK, `mcc` still compiles without errors and generates the results. But the `-R` option doesn't apply to these libraries and does not do anything.

- **-S**

  The standard behavior for the MATLAB Runtime is that every instance of a class gets its own MATLAB Runtime context. The 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 context. If multiple instances of a class are created, they use the context created by the first instance which saves startup time and some resources. However, any changes made to the global workspace or the base workspace by one instance impacts all class instances. For example, if `instance1` creates a global variable A in a singleton MATLAB Runtime, then `instance2` can 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 MATLAB Runtime. You do not need to perform other steps.
.NET assembly	Default behavior for target is singleton MATLAB Runtime. You do not need to perform other steps.
COM component Java package	• Using the Library Compiler app, click **Settings** and add `-S` to the **Additional parameters passed to MCC** field. • Using `mcc`, pass the `-S` flag.

- **-T**

Specify the output target phase and type.

Use the syntax `-T` *target* to define the output type.

Target	Description
`compile:exe`	Generate a C/C++ wrapper file, and compile C/C++ files to an object form suitable for linking into a standalone application.

Target	Description
`compile:lib`	Generate a C/C++ wrapper file, and compile C/C++ files to an object form suitable for linking into a shared library or DLL.
`link:exe`	Same as `compile:exe` and also link object files into a standalone application.
`link:lib`	Same as `compile:lib` and also link object files into a shared library or DLL.

- **-u**

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

- **-U**

  Build deployable archive (`.ctf` file) for MATLAB Production Server.

- **-v**

  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**

  Display warning messages. Use the syntax

  `-w option [:<msg>]`

  to control the display of warnings.

Syntax	Description
`-w list`	List all of the possible warnings that `mcc` can generate.
`-w enable`	Enable all warnings.
`-w disable[:<string>]`	Disable specific warnings associated with *<string>*. See "Warning Messages" (MATLAB Compiler) 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" (MATLAB Compiler) 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 off warnings for specific error messages defined by *<string>*. You can also narrow the 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.

You can also turn warnings on or off in your MATLAB code.

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

```
if isdeployed
    warning off
end
```

To turn on warnings for deployed applications, you write:

```
if isdeployed
    warning on
end
```

- **-W**

  Control the generation of function wrappers. Use the syntax

  `-W type`

  to control the generation of function wrappers for a collection of MATLAB files generated by the compiler. You provide a list of functions, and the compiler generates the wrapper functions and any appropriate global variable definitions.

- **-Y** Use

  `-Y license.lic`

  to override the default license file with the specified argument.

  Note The -Y flag works only with the command-line mode.

  `>>!mcc -m foo.m -Y license.lic`

See Also

6

Apps — Alphabetical List

Production Server Compiler

Package MATLAB programs 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" on page 2-2
- "Create and Install a Deployable Archive with Excel Integration For MATLAB Production Server"

Parameters

type — type of archive generated
Deployable Archive | Deployable Archive with Excel Integration

Type of archive to generate as a character array.

exported functions — functions to package
list of character arrays

Functions to package as a list of character arrays.

`archive information` — name of the archive
character array

Name of the archive as a character array.

`files required for your archive to run` — files that must be included with archive
list of files

Files that must be included with archive as a list of files.

`files packaged with the archive` — optional files installed with archive
list of files

Optional files installed with archive as a list of files.

Settings

`Additional parameters passed to MCC` — flags controlling the behavior of the compiler
character array

Flags controlling the behavior of the compiler as a character array.

`testing files` — folder where files for testing are stored
character array

Folder where files for testing are stored as a character array.

`end user files` — folder where files for building a custom installer are stored
character array

Folder where files for building a custom installer are stored are stored as a character array.

`packaged installers` — folder where generated installers are stored
character array

Folder where generated installers are stored as a character array.

Programmatic Use

productionServerCompiler

See Also

Topics

"Create a Deployable Archive for MATLAB Production Server" on page 2-2
"Create and Install a Deployable Archive with Excel Integration For MATLAB Production
Server"

Introduced in R2013b

Persistence

Use a Data Cache to Persist Data

Persistence provides a mechanism to cache data between calls to MATLAB code running on a server instance. A *persistence service* runs separately from the server instance and can be started and stopped manually. A *connection name* links a server instance to a persistence service. A persistence service uses a *persistence provider* to store data. Currently, Redis™ is the only supported persistence provider. The connection name is used in MATLAB application code to create a *data cache* in the linked persistence service.

Before starting a persistence service for a server instance from the system command prompt, you need to create a JSON file called `mps_cache_config` and place it in the `config` folder of the server instance.

mps_cache_config

```
{
  "Connections": {
    "<connection_name>": {
      "Provider": "Redis",
      "Host": "localhost",
      "Port": <port_number>
    }
  }
}
```

Specify the *<connection_name>* and *<port_number>* in the JSON file. You can specify multiple connections in the file `mps_cache_config`. Each connection must have a unique name and a unique (host, port) pair. If you are using the persistence service through the dashboard, the file `mps_cache_config` is automatically created in the server instance `config` folder.

Workflow to Use Persistence

Steps	Command Line	Dashboard
1. Create file mps_cache_config	Manually create a JSON file and place it in the config folder of the server instance.	Automatically created.
2. Start persistence service	Use mps-cache to start a persistence service. For testing purposes, you can create a persistence service controller object using mps.cache.control.	• Create a persistence service. • Add the persistence service to a server instance using a connection name. • Start the persistence service. • Attach the connection associated with a persistence service to a server instance.
3. Create a data cache	Use mps.cache.connect to create a data cache.	Use mps.cache.connect to create a data cache.

Example: Increment a Counter Using a Data Cache

This example shows you how to use persistence to increment a counter using a data cache. The example presents two workflows: a testing workflow that uses the MATLAB and a deployment workflow that requires an active server instance.

Testing Workflow

1 Create a persistence service that uses Redis as the persistence provider and start the service.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519)
start(ctrl)
```

2 Write MATLAB code that creates a cache and then updates a counter using the cache. Name the file myCounter.m

myCounter.m

```matlab
function x = myCounter(cacheName,connectionName)

% create a data cache
c = mps.cache.connect(cacheName,'Connection',connectionName);

% if the key 'count' doesn't exist yet, initialize it
if isKey(c,'count') == false
    put(c,'count',0)
else
    value = get(c,'count');
    % increment the counter
    put(c,'count', value+1);
end
x = get(c,'count');
```

3 Test the counter.

```matlab
for i = 1:5
    y(i) = myCounter('myCache','myRedisConnection');
end
y

y =

     1     2     3     4     5
```

Deployment Workflow

Before you deploy code that uses persistence to a server instance, start the persistence service and attach it to the server instance. You can start the persistence service from the system command line using `mps-cache` or follow the steps in the dashboard. This example assumes your server instance uses the default host and port: `localhost:9910`.

1 Package the file `myCounter.m` using the **Production Server Compiler** app or `mcc`.
2 Deploy the archive (`myCounter.ctf` file) to the server.
3 Test the counter. You can make calls to the server using the "RESTful API" from the MATLAB desktop.

```matlab
rhs = {['myCache'],['myRedisConnection']};
body = mps.json.encoderequest(rhs,'Nargout',1);

options = weboptions;
options.ContentType = 'text';
options.MediaType = 'application/json';
options.Timeout = 30;
```

```
for i = 1:5
response = webwrite('http://localhost:9910/myCounter/myCounter', body, options);
x(i) = mps.json.decoderesponse(response);
end
x = [x{:}]

X =

    1     2     3     4     5
```

As expected, the results from the testing environment workflow and the deployment environment workflow are the same.

See Also

mps.cache.Controller | mps.cache.DataCache | mps.cache.connect | mps.cache.control | mps.sync.TimedMATFileMutex | mps.sync.TimedRedisMutex | mps.sync.mutex

Persistence Functions

mps.cache.DataCache

Represent cache concept in MATLAB code

Description

`mps.cache.DataCache` represents the concept of cache in MATLAB code. It is an abstract class that serves as a superclass for each persistence provider-specific data cache class.

Currently, Redis and MATLAB are the only supported persistence providers. Therefore, the cache objects will be of type `mps.cache.RedisCache` or `mps.cache.MATFileCache`.

Creation

Create a persistence provider-specific subclass of `mps.cache.DataCache` using `mps.cache.connect`.

Properties

See provider-specific subclasses for properties.

Object Functions

mps.cache.connect	Connect to cache, or create a cache if it doesn't exist
bytes	Return the number of bytes of storage used by value stored at each key
clear	Remove all keys and values from cache
flush	Write all locally modified keys to the persistence service
get	Fetch values of keys from cache
getp	Get the value of a public cache property
isKey	Determine if the cache contains specified keys
keys	Get all keys from cache

length	Number of key-value pairs in the data cache
purge	Flush all local data to the persistence service
put	Write key-value pairs to cache
remove	Remove keys from cache
retain	Store remote keys from cache locally or return locally stored keys

Examples

Connect to a Redis Cache

Start a persistence service that uses Redis as the persistence provider. The service requires a connection name and an open port. Once the service is running, you can connect to the service using the connection name and create a cache.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
c = mps.cache.connect('myCache', 'Connection', 'myRedisConnection')


c =

RedisCache with properties:

          Host: 'localhost'
          Port: 4519
          Name: 'myCache'
    Operations: "read | write | create | update"
     LocalKeys: {}
    Connection: 'myRedisConnection'

Use getp instead of dot notation to access properties.
```

See Also
mps.cache.Controller

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

mps.cache.Controller

Manage the life cycle of a persistence service in a MATLAB testing environment

Description

`mps.cache.Controller` is used to manage the life cycle of a persistence service in a MATLAB testing environment. You can perform various actions such as starting and stopping the service using the object.

Creation

Create a `mps.cache.Controller` object using `mps.cache.control`.

Properties

ActiveConnection — Connection indicator
True | False

This property is read-only.

Indicates whether the connection to the persistence provider is active or not. The value is `True` when the persistence service is attached to the MATLAB session, otherwise it is `False`.

Example: `ActiveConnection: False`

ManageService — Service management indicator
True | False | Unknown

This property is read-only.

Indicates whether the controller object is managing the persistence service or not. `ManageService` is `True` if the persistence service is started using the controller's `start` method and `False` if the MATLAB session is attached to the persistence service using the controller's `attach` method. In all other cases, the value is set to `Unknown`.

If `ManageService` is `True`, destroying the controller object via `delete` or exiting MATLAB will stop the persistence service.

Example: `ManageService: True`

Host — Host name
character vector

This property is read-only.

Name of the system hosting the persistence service.

This property is not displayed when you create a controller that uses MATLAB as a persistence provider.

Example: `Host: 'localhost'`

Port — Port number
positive scalar

This property is read-only.

Port number for persistence service.

This property is not displayed when you create a controller that uses MATLAB as a persistence provider.

Example: `Port: 4519`

ProviderName — Name of persistence provider
`'Redis'` | `'MatlabTest'`

This property is read-only.

Name of the persistence provider.

Currently, Redis is the only supported persistence provider.

You can also use MATLAB as a persistence provider for testing purposes. If you use MATLAB as a persistence provider, the provider name is displayed as `'MatlabTest'`.

Example: `ProviderName: 'Redis'`

Example: `ProviderName: 'MatlabTest'`

ConnectionName — Name of connection
character vector | string

This property is read-only.

Name of connection to persistence service.

Example: ConnectionName: 'myRedisConnection'

Folder* — Storage folder path
character vector

This property is read-only.

Storage folder path. The folder displayed is used as a database.

* This property is displayed only when you create a controller that uses MATLAB as a
persistence provider.

Example: Folder: 'c:\tmp'

Object Functions

mps.cache.control	Create a persistence service controller object
start	Start a persistence service and attach it a to MATLAB session
stop	Stop a persistence service and detach it from a MATLAB session
restart	Restart a persistence service and attach it to a MATLAB session
attach	Connect a MATLAB session to a persistence service that is already running
detach	Disconnect MATLAB session from a persistence service that is already running
ping	Test whether the persistence service is reachable
version	Version number for persistence provider

Examples

Create a Redis Service Controller

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519)
```

```
ctrl =

  Controller with properties:

    ActiveConnection: False
       ManageService: Unknown
                Host: 'localhost'
                Port: 4519
          Operations: "read | write | create | update"
        ProviderName: 'Redis'
      ConnectionName: 'myRedisConnection'
```

Create a MATLAB Service Controller

```
mctrl = mps.cache.control('myMATFileConnection','MatlabTest','Folder','c:\tmp')

mctrl =

  Controller with properties:

    ActiveConnection: False
       ManageService: Unknown
              Folder: 'c:\tmp'
          Operations: "read | write | create | update"
        ProviderName: 'MatlabTest'
      ConnectionName: 'myMATFileConnection'
```

See Also
`mps.cache.DataCache`

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

mps.cache.connect

Connect to cache, or create a cache if it doesn't exist

Syntax

```
c = mps.cache.connect(cacheName)
c = mps.cache.connect(cacheName,'Connection',connectionName)
```

Description

`c = mps.cache.connect(cacheName)` connects to a cache when there's a single connection to a persistence service.

`c = mps.cache.connect(cacheName,'Connection',connectionName)` connects to a cache using the connection specified by `connectionName` when there are multiple connections to a persistence service.

Examples

Create a Cache When There is a Single Connection to a Persistence Service

Start a persistence service that uses Redis as the persistence provider. The service requires a connection name and an open port. Once the service is running, you can connect to the service using the connection name and create a cache.

When you have a single connection, you do not need to specify the connection name to `mps.cache.connect`.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519)
start(ctrl)
c = mps.cache.connect('myCache');

c =
```

```
RedisCache with properties:

            Host: 'localhost'
            Port: 4519
            Name: 'myCache'
      Operations: "read | write | create | update"
       LocalKeys: {}
      Connection: 'myRedisConnection'

Use getp instead of dot notation to access properties.
```

Create a Cache When There are Multiple Connections to a Persistence Service

When you have multiple connections to a persistence service, create a cache by
specifying the connection name associated with the service you want to use.

```
ctrl_1 = mps.cache.control('myRedisConnection1','Redis','Port',4519)
start(ctrl_1)
ctrl_2 = mps.cache.control('myRedisConnection2','Redis','Port',4520)
start(ctrl_2)
c = mps.cache.connect('myCache','Connection','myRedisConnection1')

c =

RedisCache with properties:

            Host: 'localhost'
            Port: 4519
            Name: 'myCache'
      Operations: "read | write | create | update"
       LocalKeys: {}
      Connection: 'myRedisConnection1'

Use getp instead of dot notation to access properties.
```

Input Arguments

cacheName — Cache name to connect to or create
character vector

Cache name to connect to or create, specified as a character vector.

Example: `'myCache'`

`connectionName` — Name of connection
character vector

Name of connection to persistence service, specified as a character vector.

Example: `'Connection','myRedisConnection'`

Output Arguments

`c` — Data cache object
persistence provider-specific data cache object

A persistence provider specific data cache object.

Currently, Redis and MATLAB are the only supported persistence providers. Therefore, the cache objects will be of type `mps.cache.RedisCache` or `mps.cache.MATFileCache`.

See Also

Introduced in R2018b

mps.cache.control

Create a persistence service controller object

Syntax

```
ctrl = mps.cache.control(connectionName,Provider,'Port',num)
ctrl = mps.cache.control(connectionName,Provider,'Folder',
folderPath)
```

Description

`ctrl = mps.cache.control(connectionName,Provider,'Port',num)` creates a
persistence service controller object using a connection to a persistence service specified
by `connectionName`, a persistence provider specified by `Provider`, and a port number
`num` for the service.

`ctrl = mps.cache.control(connectionName,Provider,'Folder',
folderPath)` creates a persistence service controller object that uses a folder specified
by `folderPath` as a database.

Use this syntax when you want to use MATLAB as a persistence provider for testing
purposes.

Examples

Create a Redis Service Controller

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519)

ctrl =

  Controller with properties:

    ActiveConnection: False
```

```
      ManageService: Unknown
               Host: 'localhost'
               Port: 4519
         Operations: "read | write | create | update"
       ProviderName: 'Redis'
     ConnectionName: 'myRedisConnection'
```

Create a MATLAB Service Controller

```
mctrl = mps.cache.control('myMATFileConnection','MatlabTest','Folder','c:\tmp')

mctrl =

  Controller with properties:

    ActiveConnection: False
       ManageService: Unknown
              Folder: 'c:\tmp'
          Operations: "read | write | create | update"
        ProviderName: 'MatlabTest'
      ConnectionName: 'myMATFileConnection'
```

Input Arguments

connectionName — Name of the connection
character vector | string

Name of the connection to the persistence service, specified as a character vector.

The connectionName links a MATLAB session to a persistence service.

Example: 'myRedisConnection'

Provider — Name of the persistence provider
'Redis' | 'MatlabTest'

Name of the persistence provider, specified as a character vector.

You can use MATLAB as a persistence provider for testing purposes. If you use MATLAB as a persistence provider, specify the provider name as 'MatlabTest'.

Example: 'Redis'

Example: `'MatlabTest'`

num — Port number
positive scalar

Port number for the persistence service.

Example: `'Port', 4519`

folderPath — Storage folder path
character vector

Storage folder path, specified as a character vector.

Specify this input only when you want to use MATLAB as a persistence provider for testing purposes. A folder specified by `folderPath` serves as a database.

Example: `'Folder','c:\tmp'`

Output Arguments

ctrl — Persistence provider service controller object
`mps.cache.Controller` object

Persistence provider service controller returned as a `mps.cache.Controller` object.

See Also

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

attach

Connect a MATLAB session to a persistence service that is already running

Syntax

```
attach(ctrl)
```

Description

`attach(ctrl)` connects a MATLAB session to a persistence service that is already running.

Examples

Connect a MATLAB Session to a Persistence Service

Attach MATLAB code to a persistence service.

Start a persistence service outside your MATLAB session from system command line using `mps-cache` or using the dashboard. Assuming your started the service using a connection name `myOutsideRedisConnection` at port `8899`, attach your MATLAB session to it from the MATLABdesktop.

```
ctrl = mps.cache.control('myOutsideRedisConnection','Redis','Port',8899);
attach(ctrl)
```

Input Arguments

ctrl — Service controller
`mps.cache.Controller` object

Persistence service controller, represented as a `mps.cache.Controller` object.

Example: `attach(ctrl)`

See Also

detach | restart | start | stop

Topics

"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

detach

Disconnect MATLAB session from a persistence service that is already running

Syntax

```
detach(ctrl)
```

Description

detach(`ctrl`) disconnects MATLAB session from a persistence service that is already running.

Examples

Disconnect MATLAB Code

Disconnect MATLAB code from a persistence service.

First, create a persistence service controller object and use that object to start the persistence service. Once you have a persistence service running, you can connect MATLAB code to it. You can then disconnect the code from the service.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
attach(ctrl)
detach(ctrl)
```

Input Arguments

ctrl — Service controller
mps.cache.Controller object

Persistence service controller, represented as a mps.cache.Controller object.

Example: `detach(ctrl)`

See Also
`attach` | `restart` | `start` | `stop`

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

start

Start a persistence service and attach it a to MATLAB session

Syntax

```
start(ctrl)
```

Description

`start(ctrl)` starts a persistence service represented by `ctrl` and attaches it to a current MATLAB session.

- To make a persistence service available in a MATLAB session, the service must be started and then attached to the MATLAB session. `start` performs both these actions.
- If a persistence service has already been started, there is no need to call `start`. Use `attach` instead.
- `start` and `stop`, `attach` and `detach` must be used in pairs.
- If you connected a persistence service to your MATLAB session with `start`, you must disconnect with `stop`.
- If you connected with `attach`, you must disconnect with `detach`.

Examples

Start a Persistence Service

Start a persistence service.

First, create a persistence service controller object and use that object to start the persistence service.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
```

Input Arguments

`ctrl` — Service controller
`mps.cache.Controller` object

Persistence service controller, represented as a `mps.cache.Controller` object.

Example: `start(ctrl)`

See Also

`attach` | `detach` | `restart` | `stop`

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

stop

Stop a persistence service and detach it from a MATLAB session

Syntax

```
stop(ctrl)
```

Description

`stop(ctrl)` stops a persistence service represented by `ctrl` and detaches it from a current MATLAB session.

- You cannot stop a service that has not been started.
- You can only stop a service that has been started using `start`.
- Exiting MATLAB will automatically call `stop` on all persistence services that were started using `start`.

Examples

Stop a Persistence Service

Stop a persistence service.

First, create a persistence service controller object and use that object to start the persistence service. Once you have a persistence service running, you can then stop it.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
stop(ctrl)
```

Input Arguments

ctrl — Service controller
mps.cache.Controller object

Persistence service controller, represented as a mps.cache.Controller object.

Example: stop(ctrl)

See Also

attach | detach | restart | start

Topics

"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

restart

Restart a persistence service and attach it to a MATLAB session

Syntax

```
restart(ctrl)
```

Description

`restart(ctrl)` restarts a persistence service represented by `ctrl`. You only restart a services you originally started using `start`.

Examples

Restart a Persistence Provider

Restart a persistence service.

First, create a persistence service controller object and use that object to start the persistence service. Once you have a persistence service running, you can then restart it.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
restart(ctrl)
```

Input Arguments

`ctrl` — Service controller
`mps.cache.Controller` object

Persistence service controller, represented as a `mps.cache.Controller` object.

Example: `restart(ctrl)`

See Also

attach | detach | start | stop

Topics

"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

ping

Test whether the persistence service is reachable

Syntax

```
ping(ctrl)
```

Description

`ping(ctrl)` tests whether the persistence service is reachable. In order to ping a persistence service, it must be started and attached to yourMATLAB session.

Examples

Ping Persistence Service

Test whether the persistence service is reachable.

First, create a persistence service controller object and use that object to start the persistence service. Once you have a persistence service running, you can ping the service.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
ping(ctrl)

Sending ping to Redis on localhost:4519.
Redis service running on localhost:4519.

ans =

  logical
```

1

Input Arguments

ctrl — Service controller
mps.cache.Controller object

Persistence service controller, represented as a mps.cache.Controller object.

Example: ping(ctrl)

See Also

restart | start | stop

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

version

Version number for persistence provider

Syntax

```
version(ctrl)
```

Description

`version(ctrl)` returns the version number for the persistence provider. In order to get the version number of the persistence provider, the persistence service must be started and attached to yourMATLAB session.

Examples

Get Version Number

Get the version number of the persistence provider that the persistence service is connected to.

First, create a persistence service controller object and use that object to start the persistence service. Once you have a persistence service running, you can get the version number.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
version(ctrl)
```

```
Redis version: 3.0.504
```

Input Arguments

ctrl — Service controller
mps.cache.Controller object

Persistence service controller, represented as a `mps.cache.Controller` object.

Example: `version(ctrl)`

See Also

restart | start | stop

Topics

"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

bytes

Return the number of bytes of storage used by value stored at each key

Syntax

```
b = bytes(c,keys)
```

Description

`b = bytes(c,keys)` returns the number of bytes of storage used by value stored at each key.

Examples

Get the Number of Bytes of Storage Used by a Value in the Cache

Start a persistence service that uses Redis as the persistence provider. The service requires a connection name and an open port. Once the service is running, you can connect to the service using the connection name and create a cache.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
c = mps.cache.connect('myCache', 'Connection', 'myRedisConnection');
```

Add keys and values to the cache and then get the number of bytes of storage used by a value stored at each key in the cache. Represent the keys and the bytes used by each value of key as a MATLAB table.

```
put (c,'keyOne',10,'keyTwo',20,'keyThree',30,'keyFour',[400 500],'keyFive',magic(5))
b = bytes(c,{'keyOne','keyTwo','keyThree','keyFour','keyFive'})
tt = table(keys(c), bytes(c,keys(c))','VariableNames',{'Keys','Bytes'})

b =

    72    72    72    80    264
```

```
tt =

  5×2 table

      Keys        Bytes

    _____   _____

    'keyFive'      264
    'keyFour'       80
    'keyOne'        72
    'keyThree'      72
    'keyTwo'        72
```

Input Arguments

c — Data cache
persistence provider specific data cache object

A data cache represented by a persistence provider specific data cache object.

Currently, Redis and MATLAB are the only supported persistence providers. Therefore, the cache objects will be of type `mps.cache.RedisCache` or `mps.cache.MATFileCache`.

Example: c

keys — Keys
cell array of character vectors

A list of all the keys, specified as a cell array of character vectors.

Example: `{'keyOne','keyTwo','keyThree','keyFour','keyFive'}`

Output Arguments

b — Number of bytes
numeric row vector

Number of bytes used by each value associated with a key, returned as a numeric row vector.

The byte counts in the output vector appear in the same order as the corresponding input keys. `b(i)` is the byte count for `keys(i)`.

See Also

get | keys | length | put

Topics

"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

clear

Remove all keys and values from cache

Syntax

```
n = clear(c)
```

Description

`n = clear(c)` removes all keys and values from cache and returns the number of keys cleared from the cache in `n`.

`clear` removes both local and remote keys and values.

Examples

Clear All Keys and Values from Cache

Start a persistence service that uses Redis as the persistence provider. The service requires a connection name and an open port. Once the service is running, you can connect to the service using the connection name and create a cache.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
c = mps.cache.connect('myCache', 'Connection', 'myRedisConnection');
```

Add keys and values to the cache and display them as a MATLAB table.

```
put(c,'keyOne',10,'keyTwo',20,'keyThree',30,'keyFour',[400 500],'keyFive',magic(5))
tt = table(keys(c), get(c,keys(c))','VariableNames',{'Keys','Values'})

tt =

  5×2 table

      Keys          Values
    _____    _____
```

```
'keyFive'      [5×5 double]
'keyFour'      [1×2 double]
'keyOne'       [        10]
'keyThree'     [        30]
'keyTwo'       [        20]
```

Clear the cache and check if it is empty.

```
n = clear(c)
k = keys(c)

n =

  int64

   5


k =

  0×1 empty cell array
```

Input Arguments

c — Data cache
persistence provider specific data cache object

A data cache represented by a persistence provider specific data cache object.

Currently, Redis and MATLAB are the only supported persistence providers. Therefore, the cache objects will be of type `mps.cache.RedisCache` or `mps.cache.MATFileCache`.

Example: c

Output Arguments

n — Number of key-value pairs
integer

Number of key-value pairs removed, returned as an integer.

Example: 5

See Also

flush | keys | purge | put | remove | retain

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

flush

Write all locally modified keys to the persistence service

Syntax

```
modKeys = flush(c)
```

Description

`modKeys = flush(c)` writes all locally modified data in `c` to the persistence service and returns a list of keys that have been modified.

`flush` does not clear the list of retained keys.

Examples

Write All Locally Modified Data to the Persistence Service

Start a persistence service that uses Redis as the persistence provider. The service requires a connection name and an open port. Once the service is running, you can connect to the service using the connection name and create a cache.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
c = mps.cache.connect('myCache', 'Connection', 'myRedisConnection');
```

Add keys and values to the cache and display them as a MATLAB table.

```
put(c,'keyOne',10,'keyTwo',20,'keyThree',30,'keyFour',[400 500],'keyFive',magic(5))
tt = table(keys(c), get(c,keys(c))','VariableNames',{'Keys','Values'})

tt =

  5×2 table

      Keys          Values
    _____    _____
```

```
'keyFive'       [5×5 double]
'keyFour'       [1×2 double]
'keyOne'        [          10]
'keyThree'      [          30]
'keyTwo'        [          20]
```

Retain a single key locally and verify that it shows up as a local key in the cache object.

```
retain(c,'keyOne')
display(c)


c =

RedisCache with properties:

          Host: 'localhost'
          Port: 4519
          Name: 'myCache'
    Operations: "read | write | create | update"
     LocalKeys: {'keyOne'}
    Connection: 'myRedisConnection'

Use getp instead of dot notation to access properties.
```

Modify the local key and flush it to the remote cache. Display the keys and values in the cache as a MATLAB table.

```
put(c,'keyOne',rand(3))
modKeys = flush(c)
tt = table(keys(c), get(c,keys(c))','VariableNames',{'Keys','Values'})

modKeys =

  1×1 cell array

    {'keyOne'}

tt =

  5×2 table

      Keys           Values
```

```
     _____      _____

'keyFive'      [5×5 double]
'keyFour'      [1×2 double]
'keyOne'       [3×3 double]
'keyThree'     [        30]
'keyTwo'       [        20]
```

Input Arguments

c — Data cache
persistence provider specific data cache object

A data cache represented by a persistence provider specific data cache object.

Currently, Redis and MATLAB are the only supported persistence providers. Therefore, the cache objects will be of type `mps.cache.RedisCache` or `mps.cache.MATFileCache`.

Example: `c`

Output Arguments

modKeys — Modified keys
cell array of character vectors

A list of the modified keys that were written to the persistence service, returned as a cell array of character vectors.

See Also
`clear` | `keys` | `purge` | `remove` | `retain`

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

get

Fetch values of keys from cache

Syntax

```
values = get(c,keys)
```

Description

`values = get(c,keys)` fetches values of keys specified by `keys` from the cache specified by `c`. Values are returned in the same order as input variables as a cell array.

Examples

Get Values for Keys from Cache

Start a persistence service that uses Redis as the persistence provider. The service requires a connection name and an open port. Once the service is running, you can connect to the service using the connection name and create a cache.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
c = mps.cache.connect('myCache', 'Connection', 'myRedisConnection');
```

Add keys and values to the cache.

```
put(c,'keyOne',10,'keyTwo',20,'keyThree',30,'keyFour',[400 500],'keyFive',magic(5))
```

Get all the keys and associated values and display them as a MATLAB table.

```
k = keys(c)
v = get(c,{'keyOne','keyTwo','keyThree','keyFour','keyFive'})
tt = table(keys(c), get(c,keys(c))','VariableNames',{'Keys','Values'})

k =

  5×1 cell array
```

```
    {'keyFive' }
    {'keyFour' }
    {'keyOne'  }
    {'keyThree'}
    {'keyTwo'  }


v =

  1×5 cell array

    {[10]}    {[20]}    {[30]}    {1×2 double}    {5×5 double}


tt =

  5×2 table

      Keys          Values
    _____    _____

    'keyFive'     [5×5 double]
    'keyFour'     [1×2 double]
    'keyOne'      [        10]
    'keyThree'    [        30]
    'keyTwo'      [        20]
```

Input Arguments

c — Data cache
persistence provider specific data cache object

A data cache represented by a persistence provider specific data cache object.

Currently, Redis and MATLAB are the only supported persistence providers. Therefore, the cache objects will be of type `mps.cache.RedisCache` or `mps.cache.MATFileCache`.

Example: c

keys — Keys
cell array of character vectors

A cell array of keys whose values you want to retrieve from cache.

Example: `{'keyOne','keyTwo','keyThree','keyFour','keyFive'}`

Output Arguments

`values` — Values
cell array

A list of values associated with keys, returned as a cell array.

See Also
`getp` | `keys` | `length` | `put`

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

getp

Get the value of a public cache property

Syntax

```
value = getp(c,property)
```

Description

`value = getp(c,property)` gets the value of a public cache property.

Ordinarily, you would be able to access the public properties of a cache object using the dot notation. For example: `c.Connection`. However, all cache objects use dot reference and dot assignment to refer to keys stored in the cache rather than cache object properties. Therefore, `c.Connection` refers to a key named `Connection` in the cache instead of the cache's `Connection` property.

There is no `setp` method since all cache properties are read-only.

Examples

Get the Value of a Named, Public, Hidden Property

Start a persistence service that uses Redis as the persistence provider. The service requires a connection name and an open port. Once the service is running, you can connect to the service using the connection name and create a cache.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
c = mps.cache.connect('myCache', 'Connection', 'myRedisConnection');
```

Retrieve the connection name.

```
getp(c,'Connection')
```

8-41

```
ans =

    'myRedisConnection'
```

Input Arguments

`c` — Data cache
persistence provider specific data cache object

A data cache represented by a persistence provider specific data cache object.

Currently, Redis and MATLAB are the only supported persistence providers. Therefore, the cache objects will be of type `mps.cache.RedisCache` or `mps.cache.MATFileCache`.

Example: `c`

`property` — Property name
character vector

Property name, specified as a character vector. The common public cache properties are `Name`, `LocalKeys`, and `Connection`. Provider-specific cache objects may have additional properties. For example, `mps.cache.RedisCache` has the properties `Host` and `Port`.

Example: `'Connection'`

Output Arguments

`value` — Property value
valid value

A valid property value.

See Also
`get` | `keys` | `put`

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

isKey

Determine if the cache contains specified keys

Syntax

```
TF = isKey(c,keys)
```

Description

`TF = isKey(c,keys)` returns a logical `1 (true)` if `c` contains the specified key, and returns a logical `0 (false)` otherwise.

If `keys` is an array that specifies multiple keys, then `TF` is a logical array of the same size, and `TF{i}` is `true` if `keys{i}` exists in cache `c`.

Examples

Determine if the Cache Contains Specified Keys

Start a persistence service that uses Redis as the persistence provider. The service requires a connection name and an open port. Once the service is running, you can connect to the service using the connection name and create a cache.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
c = mps.cache.connect('myCache', 'Connection', 'myRedisConnection');
```

Add keys and values to the cache.

```
put(c,'keyOne',10,'keyTwo',20,'keyThree',30,'keyFour',[400 500],'keyFive',magic(5))
```

Determine if the cache contains specified keys.

```
TF = isKey(c,{'keyOne','keyTW00','keyTREE','key4','keyFive'})

TF =
```

```
1×5 logical array

 1   0   0   0   1
```

Input Arguments

c — Data cache
persistence provider specific data cache object

A data cache represented by a persistence provider specific data cache object.

Currently, Redis and MATLAB are the only supported persistence providers. Therefore, the cache objects will be of type `mps.cache.RedisCache` or `mps.cache.MATFileCache`.

Example: `c`

keys — Keys to search for
character vector | string | cell array of character vectors or strings

Keys to search for in the cache object `c`, specified as a character vector, string, or cell array of character vectors or strings. To search for multiple keys, specify `keys` as a cell array.

Example: `{'keyOne','keyTW00','keyTREE','key4','keyFive'}`

Output Arguments

TF — Logical value
logical array

A logical array of the same size as `keys` indicating which specified keys were found in the data cache. `TF` has a logical `1` (`true`) if `c` contains a key specified by `keys`, and a logical `0` (`false`) otherwise.

See Also
`get` | `keys` | `length` | `put`

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

keys

Get all keys from cache

Syntax

```
k = keys(c)
```

Description

`k = keys(c)` returns a list of all the keys in a data cache as a cell array.

Examples

Get Keys from Cache

Start a persistence service that uses Redis as the persistence provider. The service requires a connection name and an open port. Once the service is running, you can connect to the service using the connection name and create a cache.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
c = mps.cache.connect('myCache', 'Connection', 'myRedisConnection');
```

Add keys and values to the cache.

```
put(c,'keyOne',10,'keyTwo',20,'keyThree',30,'keyFour',[400 500],'keyFive',magic(5))
```

Get all keys.

```
k = keys(c)

k =

  5×1 cell array

    {'keyFive' }
    {'keyFour' }
```

```
{'keyOne'  }
{'keyThree'}
{'keyTwo'  }
```

Input Arguments

c — Data cache
persistence provider specific data cache object

A data cache represented by a persistence provider specific data cache object.

Currently, Redis and MATLAB are the only supported persistence providers. Therefore, the cache objects will be of type `mps.cache.RedisCache` or `mps.cache.MATFileCache`.

Example: `c`

Output Arguments

k — Keys
cell array of character vectors

Keys from cache, returned as a cell array of character vectors.

See Also
bytes | get | isKey | length | put

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

length

Number of key-value pairs in the data cache

Syntax

```
num = length(c)
num = length(c,location)
```

Description

`num = length(c)` returns the total number of key-value pairs in the data cache `c`.

`num = length(c,location)` returns the numbers of key-value pairs in the data cache `c` stored remotely or locally as specified by `location`.

Examples

Count the Number of Key-Value Pairs

Start a persistence service that uses Redis as the persistence provider. The service requires a connection name and an open port. Once the service is running, you can connect to the service using the connection name and create a cache.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
c = mps.cache.connect('myCache', 'Connection', 'myRedisConnection');
```

Retain a few keys locally.

```
retain(c, {'keyOne','keyTwo'})
```

Add keys and values to the cache.

```
put(c,'keyOne',10,'keyTwo',20,'keyThree',30,'keyFour',[400 500],'keyFive',magic(5))
```

Count the number of keys-value pairs.

```
numTotal = length(c)
numRemote = length(c,'Remote')
numLocal = length(c,'Local')

numTotal =

  int64

   5

numRemote =

  int64

   3

numLocal =

  int64

   2
```

Since `keyOne` and `keyTwo` were retained before being written to the cache, they were never written to the persistence service. They are stored locally until flushed or purged to the persistence service.

Input Arguments

c — Data cache
persistence provider specific data cache object

A data cache represented by a persistence provider specific data cache object.

Currently, Redis and MATLAB are the only supported persistence providers. Therefore, the cache objects will be of type `mps.cache.RedisCache` or `mps.cache.MATFileCache`.

Example: c

location — Location name
'Remote' | 'Local'

Location of keys specified as an enumerated member of the class `mps.cache.Location`. The valid location options are either `'Remote'` or `'Local'`.

Example: `'Remote'`

Output Arguments

num — Number of keys
integer

Total number of key-value pairs in the data cache or the number stored remotely or locally, returned as an integer.

See Also
bytes | get | isKey | keys | put

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

countRemoteKeys

Count the number of keys stored on a remote persistence provider

Syntax

```
count = countRemoteKeys(c)
```

Description

`count = countRemoteKeys(c)` counts the number of keys stored on a remote persistence provider.

Examples

Count the Number of Keys Stored on a Remote Persistence Provider

```
count = countRemoteKeys(c)
```

Input Arguments

c — Data cache object
`mps.cache.DataCache` object

Example:

Output Arguments

count —

See Also

Introduced in R2018b

purge

Flush all local data to the persistence service

Syntax

```
purgedKeys = purge(c)
```

Description

`purgedKeys = purge(c)` flushes all local data to the persistence service and removes it locally.

Examples

Flush All Local Data to the Persistence Service

Start a persistence service that uses Redis as the persistence provider. The service requires a connection name and an open port. Once the service is running, you can connect to the service using the connection name and create a cache.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
c = mps.cache.connect('myCache', 'Connection', 'myRedisConnection');
```

Add keys and values to the cache.

```
put(c,'keyOne',10,'keyTwo',20,'keyThree',30,'keyFour',[400 500],'keyFive',magic(5))
```

Retain a few keys locally. For more information, see `retain`.

```
retain(c, {'keyOne','keyTwo'})
```

Modify the local keys and purge the data. Display the keys and values in the cache as a MATLAB table.

```
put(c,'keyOne',rand(3),'keyTwo', eye(10))
purgedKeys = purge(c)
```

```
tt = table(keys(c), get(c,keys(c))','VariableNames',{'Keys','Values'})
display(c)

purgedKeys =

  2×1 cell array

    {'keyOne'}
    {'keyTwo'}


tt =

  5×2 table

      Keys            Values
    _____    _____

    'keyFive'     [ 5×5  double]
    'keyFour'     [ 1×2  double]
    'keyOne'      [ 3×3  double]
    'keyThree'    [          30]
    'keyTwo'      [10×10 double]


c =

RedisCache with properties:

          Host: 'localhost'
          Port: 4519
          Name: 'myCache'
    Operations: "read | write | create | update"
     LocalKeys: {}
    Connection: 'myRedisConnection'

Use getp instead of dot notation to access properties.
```

Input Arguments

c — Data cache
persistence provider specific data cache object

A data cache represented by a persistence provider specific data cache object.

Currently, Redis and MATLAB are the only supported persistence providers. Therefore, the cache objects will be of type `mps.cache.RedisCache` or `mps.cache.MATFileCache`.

Example: `c`

Output Arguments

purgedKeys — Purged keys
cell array of character vectors

List of keys that were written to the persistence service, returned as a cell array of character vectors.

See Also
bytes | clear | flush | get | getp | isKey | keys | length | purge | put | remove | retain

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

put

Write key-value pairs to cache

Syntax

```
put(c,key1,value1,...,keyN,valueN)
put(c,keySet,valueSet)
```

Description

put(c,key1,value1,...,keyN,valueN) writes key-value pairs to cache. You can store any type of MATLAB data in a cache.

put(c,keySet,valueSet) writes key-value pairs to cache with keys from by keySet, each mapped to a corresponding value from valueSet. The input arguments keySet and valueSet must have the same number of elements, with keySet having elements that are unique.

Examples

Write a Series of Key-Value Pairs to Cache

Start a persistence service that uses Redis as the persistence provider. The service requires a connection name and an open port. Once the service is running, you can connect to the service using the connection name and create a cache.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
c = mps.cache.connect('myCache', 'Connection', 'myRedisConnection');
```

Add keys and values to the cache and display them as a MATLAB table.

```
put(c,'keyOne',10,'keyTwo',20,'keyThree',30,'keyFour',[400 500],'keyFive',magic(5))
tt = table(keys(c), get(c,keys(c))','VariableNames',{'Keys','Values'})

tt =
```

```
5×2 table

    Keys            Values
 _____      _____

'keyFive'       [5×5 double]
'keyFour'       [1×2 double]
'keyOne'        [        10]
'keyThree'      [        30]
'keyTwo'        [        20]
```

Write a Set of Keys and Corresponding Values to Cache

Start a persistence service that uses Redis as the persistence provider. The service requires a connection name and an open port. Once the service is running, you can connect to the service using the connection name and create a cache.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
c = mps.cache.connect('myCache', 'Connection', 'myRedisConnection');
```

Add a set of keys and corresponding values to the cache and display them as a MATLAB table.

```
keySet = {'keyOne','keyTwo','keyThree','keyFour','keyFive'}
valueSet = {10, 20, 30, [400 500], magic(5)}
put(d,keySet,valueSet)
tt = table(keys(c), get(c,keys(c))','VariableNames',{'Keys','Values'})
```

```
tt =

  5×2 table

    Keys            Values
 _____      _____

'keyFive'       [5×5 double]
'keyFour'       [1×2 double]
'keyOne'        [        10]
```

```
'keyThree'    [        30]
'keyTwo'      [        20]
```

Input Arguments

c — Data cache
persistence provider specific data cache object

A data cache represented by a persistence provider specific data cache object.

Currently, Redis and MATLAB are the only supported persistence providers. Therefore, the cache objects will be of type `mps.cache.RedisCache` or `mps.cache.MATFileCache`.

Example: c

key — Key
character vector

Key to add, specified as a character vector.

Example: `'keyFour'`

value — Value
array

Value, specified as an array. `value` can be any valid MATLAB data type.

Example: `[400, 500]`

keySet — Keys
cell array of character vectors

Keys, specified as a cell array of character vectors.

Example: `{'keyOne','keyTwo','keyThree','keyFour','keyFive'}`

valueSet — Values
cell array

Values, specified as comma-separated cell array. Each value may be any valid MATLAB data type.

Example: `{10, 20, 30, [400 500], magic(5)}`

See Also

bytes | clear | get | keys | length | remove

Topics

"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

remove

Remove keys from cache

Syntax

```
num = remove(c,keys)
```

Description

`num = remove(c,keys)` removes keys and associated values from cache. There is no way to recover removed keys.

Examples

Remove Keys from Cache

Start a persistence service that uses Redis as the persistence provider. The service requires a connection name and an open port. Once the service is running, you can connect to the service using the connection name and create a cache.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
c = mps.cache.connect('myCache', 'Connection', 'myRedisConnection');
```

Add keys and values to the cache and display them as a MATLAB table.

```
put(c,'keyOne',10,'keyTwo',20,'keyThree',30,'keyFour',[400 500],'keyFive',magic(5))
tt = table(keys(c), get(c,keys(c))','VariableNames',{'Keys','Values'})

tt =

  5×2 table

      Keys           Values
    _____     _____

    'keyFive'      [5×5 double]
```

```
'keyFour'      [1×2 double]
'keyOne'       [        10]
'keyThree'     [        30]
'keyTwo'       [        20]
```

Remove two keys from cache `c` and display the remaining keys and values in the cache as a MATLAB table.

```
num = remove(c,{'keyThree','keyFour'})
tt = table(keys(c), get(c,keys(c))','VariableNames',{'Keys','Values'})

num =

  int64

   2


tt =

  3×2 table

     Keys            Values
    _____      _____

    'keyFive'     [5×5 double]
    'keyOne'      [        10]
    'keyTwo'      [        20]
```

Input Arguments

c — Data cache
persistence provider specific data cache object

A data cache represented by a persistence provider specific data cache object.

Currently, Redis and MATLAB are the only supported persistence providers. Therefore, the cache objects will be of type `mps.cache.RedisCache` or `mps.cache.MATFileCache`.

Example: c

keys — Keys to remove
cell array of character vectors

Keys to remove from cache, specified as a cell array of character vectors.

Example: `{'keyThree','keyFour'}`

Output Arguments

num — Number of keys removed
integer

Number of keys removed, returned as an integer.

See Also

`clear` | `get` | `keys` | `purge` | `put` | `retain`

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

retain

Store remote keys from cache locally or return locally stored keys

Syntax

```
retain(c,remoteKeys)
localKeys = retain(c)
```

Description

`retain(c,remoteKeys)` stores keys from cache locally.

`localKeys = retain(c)` returns a cell array of keys stored locally.

Examples

Store Keys from Cache Locally and Check Local Keys

Start a persistence service that uses Redis as the persistence provider. The service requires a connection name and an open port. Once the service is running, you can connect to the service using the connection name and create a cache.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
c = mps.cache.connect('myCache', 'Connection', 'myRedisConnection');
```

Add keys and values to the cache.

```
put(c,'keyOne',10,'keyTwo',20,'keyThree',30,'keyFour',[400 500],'keyFive',magic(5))
```

Retain a few keys locally and check local keys.

```
retain(c,{'keyThree','keyFour'})
localKeys = retain(c)

localKeys =
```

```
  1×2 cell array

    {'keyThree'}    {'keyFour'}
```

Input Arguments

c — Data cache
persistence provider specific data cache object

A data cache represented by a persistence provider specific data cache object.

Currently, Redis and MATLAB are the only supported persistence providers. Therefore, the cache objects will be of type `mps.cache.RedisCache` or `mps.cache.MATFileCache`.

Example: `c`

`remoteKeys` — Keys
cell array of character vectors

Remote keys to store locally, specified as a cell array of character vectors.

Example: `{'keyThree','keyFour'}`

Output Arguments

`localKeys` — Keys
cell array of character vectors

Locally stored keys, returned as a cell array of character vectors.

Tips

- As a performance optimization you may choose to temporarily store a set of keys and their values in your MATLAB session or worker instead of the persistence service. Keys *retained* in the this fashion will be automatically written to the persistence service (see `flush`) when MATLAB exits or when the first function call returns.
- Manually control the lifetime of retained keys with the `flush` and `purge` methods.

See Also

clear | flush | purge | remove

Topics

"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

mps.sync.mutex

Create a persistence service mutex

Syntax

```
lk = mps.sync.lock(mutexName,'Connection',connectionName,Name,Value)
```

Description

`lk = mps.sync.lock(mutexName,'Connection',connectionName,Name,Value)`
creates a database advisory lock object.

Examples

Create a Redis Mutex

First, create a persistence service controller object and use that object to start the
persistence service.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
```

Use the connection name to create a persistence service mutex.

```
lk = mps.sync.mutex('myMutex','Connection','myRedisConnection')

lk =

  TimedRedisMutex with properties:

        Expiration: 10
```

```
ConnectionName: 'myRedisConnection'
      MutexName: 'myMutex'
```

Input Arguments

mutexName — Mutex name
character vector

Name of persistence service mutex, specified as a character vector.

Example: `'myMutex'`

connectionName — Name of connection
character vector

Name of connection to persistence service, specified as a character vector.

Example: `'Connection','myRedisConnection'`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example:

Expiration — Time in seconds
positive integer

Expiration time in seconds after the lock is acquired.

Other clients will be able to acquire the lock even if you do not release it.

Example: `'Expiration', 10`

Output Arguments

lk — Mutex object
persistence service mutex object

A persistence service mutex object. If you use Redis as your persistence provider, `lk` will be a `mps.sync.TimedRedisMutex` object. If you use MATLAB as your persistence provider, `lk` will be a `mps.sync.TimedMATFileMutex` object.

Tips

- A persistence service mutex allows multiple clients to take turns using a shared resource. Each cooperating client creates a mutex object with the same name using a connection to a shared persistence service. To gain exclusive access to the shared resource, a client attempts to acquire a lock on the mutex. When the client finishes operating on the shared resource, it releases the lock. To prevent lockouts should the locking client crash, all locks expire after a certain amount of time.

- Acquiring a lock on a mutex prevents other clients from acquiring a lock on that mutex but it does not lock the persistence service or any keys or values stored in the persistence service. These locks are advisory only and are meant to be used by cooperating clients intent of preventing data corruption. Rogue clients will be able to corrupt or delete data if they do not voluntarily respect the mutex locks.

See Also

acquire | mps.sync.TimedMATFileMutex | mps.sync.TimedRedisMutex | own | release

Topics

"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

mps.sync.TimedRedisMutex

Represent a Redis persistence service mutex

Description

`mps.sync.TimedRedisMutex` is a synchronization primitive used to protect data in a Redis persistence service from being simultaneously accessed by multiple workers.

Creation

Create a `mps.sync.TimedRedisMutex` object using `mps.sync.mutex`.

Properties

`Expiration` — Duration of lock in seconds
positive integer

This property is read-only.

Duration of advisory lock in seconds.

Example: 10

`ConnectionName` — Name of connection
character vector

This property is read-only.

Name of connection to persistence service.

Example: `'myRedisConnection'`

`MutexName` — Name of mutex
character vector

This property is read-only.

Name of mutex, returned as a character vector.

Example: `'myMutex'`

Object Functions

mps.sync.mutex	Create a persistence service mutex
acquire	Acquire advisory lock on persistence service mutex
own	Check ownership of advisory lock on a persistence service mutex object
release	Release advisory lock on persistence service mutex

Examples

Create a Redis Lock Object

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
lk = mps.sync.mutex('myMutex','Connection','myRedisConnection')

lk =

  TimedRedisMutex with properties:

        Expiration: 10
    ConnectionName: 'myRedisConnection'
         MutexName: 'myMutex'
```

See Also

acquire | mps.sync.TimedMATFileMutex | mps.sync.mutex | own | release

Topics

"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

mps.sync.TimedMATFileMutex

Represent a MAT-file persistence service mutex

Description

`mps.sync.TimedMATFileMutex` is synchronization primitive used to protect data in a MAT-file database from being simultaneously accessed by multiple workers.

Creation

Create a `mps.sync.TimedMATFileMutex` object using `mps.sync.mutex`.

Properties

`Expiration` — Duration of lock in seconds
positive integer

This property is read-only.

Duration of advisory lock in seconds.

Example: `10`

`ConnectionName` — Name of connection
character vector

This property is read-only.

Name of connection to persistence service.

Example: `'myRedisConnection'`

`MutexName` — Name of lock
character vector

This property is read-only.

Name of advisory lock, specified as a character vector.

Example: `'myMutex'`

Object Functions

mps.sync.mutex	Create a persistence service mutex
acquire	Acquire advisory lock on persistence service mutex
own	Check ownership of advisory lock on a persistence service mutex object
release	Release advisory lock on persistence service mutex

Examples

Create a MAT-File Lock Object

```
mctrl = mps.cache.control('myMATFileConnection','MatlabTest','Folder','c:\tmp')
start(mctrl)
lk = mps.sync.lock('myMATFileMutex','Connection','myMATFileConnection')

lk =

  TimedMATFileMutex with properties:

        Expiration: 10
    ConnectionName: 'myMATFileConnection'
         MutexName: 'myMATFileMutex'
```

See Also

acquire | mps.sync.TimedRedisMutex | mps.sync.mutex | own | release

Topics

"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

acquire

Acquire advisory lock on persistence service mutex

Syntax

```
TF = acquire(lk,timeout)
```

Description

`TF = acquire(lk,timeout)` acquires an advisory lock and returns a logical `1 (true)` if the lock was successful, and a logical `0 (false)` otherwise. If the lock is unavailable, `acquire` will continue trying to acquire it for `timeout` seconds.

Examples

Apply Advisory Lock

First, create a persistence service controller object and use that object to start the persistence service.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
```

Use the connection name to create a persistence service mutex.

```
lk = mps.sync.lock('myDbLock','Connection','myRedisConnection')
```

Try to acquire advisory lock. If lock is unavailable, retry acquiring for 20 seconds.

```
acquire(lk, 20);

TF =

  logical
```

1

Input Arguments

lk — Mutex object
persistence service mutex object

A persistence service specific mutex object. If you use Redis as your persistence provider, `lk` will be a `mps.sync.TimedRedisMutex` object. If you use a MATLAB as your persistence provider, `lk` will be a `mps.sync.TimedMATFileMutex` object.

Output Arguments

TF — Logical value
logical array

TF has a logical `1` (`true`) if acquiring the advisory lock was successful, and a logical `0` (`false`) otherwise.

See Also

`mps.sync.TimedMATFileMutex` | `mps.sync.TimedRedisMutex` | `mps.sync.mutex` | `own` | `release`

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

own

Check ownership of advisory lock on a persistence service mutex object

Syntax

```
TF = own(lk)
```

Description

`TF = own(lk)` returns a logical `1` (`true`) if you own an advisory lock on the persistence service mutex, and returns a logical `0` (`false`) otherwise.

Examples

Check If You Own the Advisory Lock

First, create a persistence service controller object and use that object to start the persistence service.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
```

Use the connection name to create a persistence service mutex.

```
lk = mps.sync.lock('myDbLock','Connection','myRedisConnection')
```

Check if you own the advisory lock.

```
TF = own(lk)

TF =

  logical
```

```
0
```

Input Arguments

lk — Mutex object
persistence service mutex object

A persistence service specific mutex object. If you use Redis as your persistence provider, lk will be a `mps.sync.TimedRedisMutex` object. If you use a MATLAB as your persistence provider, lk will be a `mps.sync.TimedMATFileMutex` object.

Output Arguments

TF — Logical value
logical array

TF has a logical `1 (true)` if you own the advisory lock on the persistence service mutex, and a logical `0 (false)` otherwise.

See Also

acquire | mps.sync.TimedMATFileMutex | mps.sync.TimedRedisMutex | mps.sync.mutex | release

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b

release

Release advisory lock on persistence service mutex

Syntax

```
TF = release(lk)
```

Description

`TF = release(lk)` releases an advisory lock on a persistence service mutex. If the lock expires before you release it, `release` returns a logical `0 (false)`. If this occurs, it may indicate potential data corruption.

Examples

Release Advisory Lock

First, create a persistence service controller object and use that object to start the persistence service.

```
ctrl = mps.cache.control('myRedisConnection','Redis','Port',4519);
start(ctrl)
```

Use the connection name to create a persistence service mutex.

```
lk = mps.sync.lock('myDbLock','Connection','myRedisConnection')
```

Try to acquire advisory lock. If lock is unavailable, retry acquiring for 20 seconds.

```
acquire(lk, 20);
```

Release lock.

```
TF = release(lk)
```

```
TF =

  logical

   1
```

Input Arguments

`lk` — Mutex object
persistence service mutex object

A persistence service specific mutex object. If you use Redis as your persistence provider, `lk` will be a `mps.sync.TimedRedisMutex` object. If you use a MATLAB as your persistence provider, `lk` will be a `mps.sync.TimedMATFileMutex` object.

Output Arguments

TF — Logical value
logical array

TF has a logical `1` (`true`) if releasing the advisory lock was successful, and a logical `0` (`false`) otherwise.

See Also

`acquire` | `mps.sync.TimedMATFileMutex` | `mps.sync.TimedRedisMutex` | `mps.sync.mutex` | `own`

Topics
"Use a Data Cache to Persist Data" on page 7-2

Introduced in R2018b