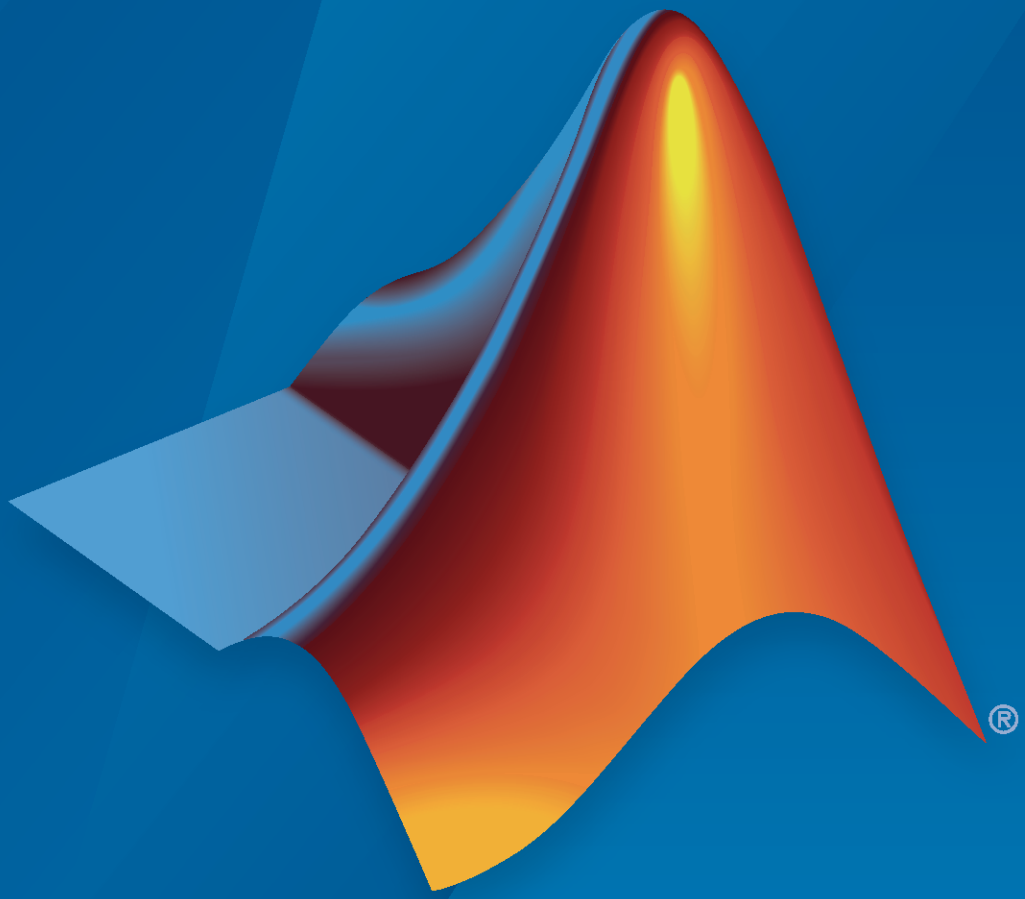
MATLAB® Production Server™

Getting Started



MATLAB®

R2020b

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.
1 Apple Hill Drive
Natick, MA 01760-2098

MATLAB® Production Server™ Getting Started Guide

Trademarks

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

Patents

MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

Revision History

September 2012	Online only	New for Version 1.0 (Release R2012b)
March 2013	Online only	Revised for Version 1.0.1 (Release R2013a)
October 2013	Online only	Revised for Version 1.1 (Release R2013b)
March 2014	Online only	Revised for Version 1.2 (Release R2014a)
October 2014	Online only	Revised for Version 2.0 (Release R2014b)
March 2015	Online only	Revised for Version 2.1 (Release R2015a)
September 2015	Online only	Revised for Version 2.2 (Release R2015b)
March 2016	Online only	Revised for Version 2.3 (Release 2016a)
September 2016	Online only	Revised for Version 2.4 (Release 2016b)
March 2017	Online only	Revised for Version 3.0 (Release 2017a)
September 2017	Online only	Revised for Version 3.0.1 (Release R2017b)
March 2018	Online only	Revised for Version 3.1 (Release R2018a)
September 2018	Online only	Revised for Version 4.0 (Release R2018b)
March 2019	Online only	Revised for Version 4.1 (Release R2019a)
September 2019	Online only	Revised for Version 4.2 (Release R2019b)
March 2020	Online only	Revised for Version 4.3 (Release R2020a)
September 2020	Online only	Revised for Version 4.4 (Release R2020b)

Contents

1 Overview

2 Installation

3 Set Up

4 Licensing

5 Deploying an Application

Overview

MATLAB Production Server Product Description

Integrate MATLAB algorithms into web, database, and enterprise applications

MATLAB Production Server lets you incorporate custom analytics into web, database, and production enterprise applications running on dedicated servers or in the cloud. You can create algorithms in MATLAB, package them using MATLAB Compiler SDK™, and then deploy them to MATLAB Production Server without recoding or creating custom infrastructure. Users can then access the latest version of your analytics automatically.

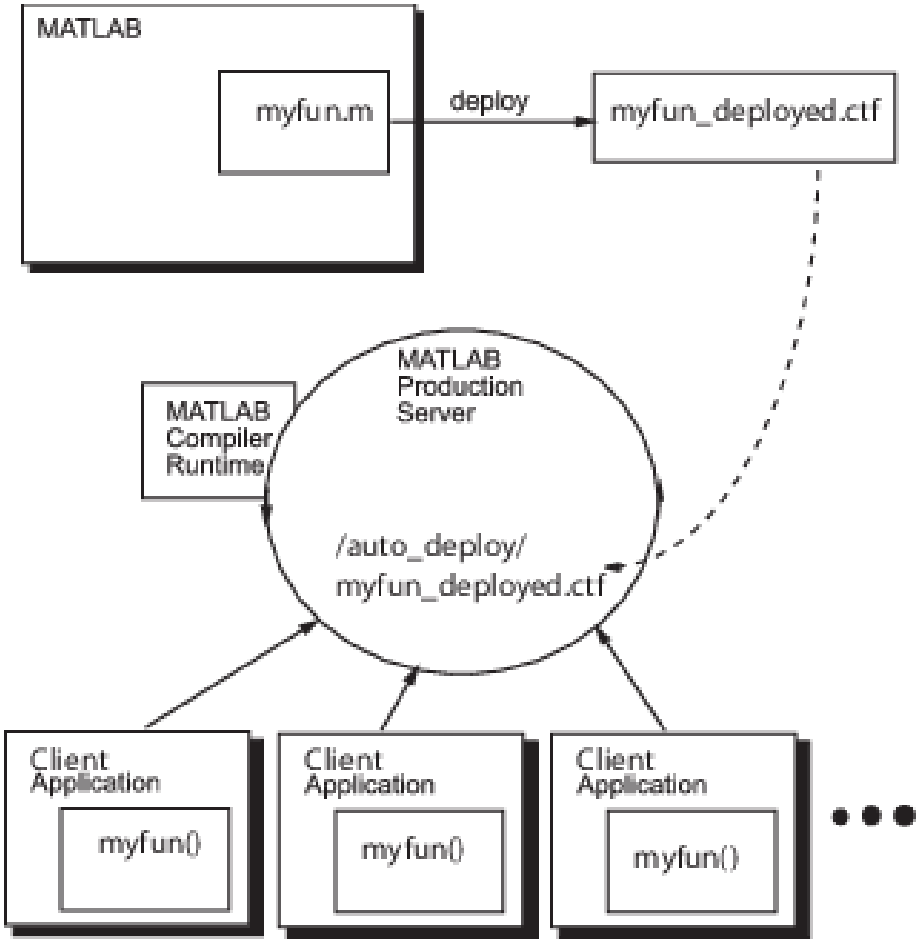
MATLAB Production Server manages multiple MATLAB Runtime versions simultaneously. As a result, algorithms developed in different versions of MATLAB can be incorporated into your application. The server runs on multiprocessor and multicore computers, providing low-latency processing of concurrent work requests. You can deploy the server on additional computing nodes to scale capacity and provide redundancy.

Key Features

- Production deployment of MATLAB programs without recoding or creating custom infrastructure
- Scalable performance and management of MATLAB analytics and MATLAB Runtime versions
- Lightweight client library for secure access to analytics by enterprise applications
- Centralized analytic service accessible via the RESTful JSON interface or from .NET, Java®, C/C++, and Python® environments
- Web-based management dashboard for IT configuration and control

MATLAB Production Server Workflow

The following figure illustrates the basic workflow to deploy MATLAB code using MATLAB Production Server.



Deploying MATLAB code using MATLAB Production Server is a four-phase process:

1. Create deployable archives.

   MATLAB users write MATLAB functions and compile them into deployable archives using MATLAB Compiler™ and MATLAB Compiler SDK.
2. Deploying the archives to an instance of the MATLAB Production Server.

   Server administrators take the deployable archives and deploy them into one or more instances of the MATLAB Production Server. In addition to adding the archive to a server's deployment folder, the server administrator might need to:

   - Install a server instance.
   - Set up licenses for a server instance.
   - Configure a server instance.
   - Install a MATLAB Runtime into a server instance.

3. Write client applications that use deployed MATLAB code on the server.

   Application developers use MATLAB Production Server client APIs to write applications that use MATLAB code.
4. Install client applications on end-user computers.

   Application installers distribute the client applications to the end-users.

See Also

More About

- “Install and Start MATLAB Production Server” on page 5-5
- “Server Overview”

Installation

Install MATLAB Production Server

Installing MATLAB Production Server is similar to installing other MathWorks®products.

1. Obtain a software license by purchasing the product or requesting a trial.
2. Download the installer from MathWorks Downloads.
3. Run the installer. Select MATLAB Production Server when the installer prompts you to select a product.

   For standard installation, see “Install Products Using Internet Connection”. For installation without an internet connection, see “Install Products Using File Installation Key”.

Once the installation is complete, you can add the *$MPS_INSTALL*`\script` folder to your system `PATH` environment variable, where *$MPS_INSTALL* represents your MATLAB Production Server installation folder. This allows you to execute MATLAB Production Server commands such as `mps-new` at the command prompt without having to provide the full path to the `script` folder.

To use MATLAB Production Server, you must add the MATLAB Production Server license file to the network license manager. For more information, see “Install Network License Manager” on page 2-2.

To configure licenses for use on cloud platforms, see “Choose an Option to Activate Your License” (Licensing on Cloud Platforms).

Install Network License Manager

The network license manager manages software licenses for MathWorks products. You typically install the license manager and MATLAB Production Server on different machines. To use MATLAB Production Server, you must add your MATLAB Production Server license file to the license manager.

To install the network license manager, select one of the following installation options.

- “Install Network License Manager with Internet Connection”
- “Install Network License Manager Offline”

If your organization already has a network license manager installed, simply add the MATLAB Production Server license file to it.

See Also

`mps-setup` | `mps-start`

More About

- “Install and Start MATLAB Production Server” on page 5-5

Supported MATLAB Runtime Versions

MATLAB Runtime is a standalone set of shared libraries that enable the execution of compiled MATLAB applications or components on computers that do not have MATLAB installed. MATLAB Production Server requires a MATLAB Runtime instance to execute the deployed MATLAB applications that it hosts. If your on-premise server machine does not have MATLAB Runtime installed, download and install the MATLAB Runtime from `https://www.mathworks.com/products/compiler/mcr`. For a server deployment on the Cloud, the deployment provides the supported MATLAB Runtime versions. For more information about MATLAB Runtime, see “MATLAB Runtime” (MATLAB Compiler).

An installation of MATLAB Production Server supports MATLAB Runtime versions up to six releases back. The following table lists several MATLAB Production Server releases and the corresponding MATLAB Runtime versions that each release supports.

MATLAB Production Server	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime
MATLAB Production Server R2017b	R2015a	R2015b	R2016a	R2016b	R2017a	R2017b					
MATLAB Production Server R2018a		R2015b	R2016a	R2016b	R2017a	R2017b	R2018a				
MATLAB Production Server R2018b			R2016a	R2016b	R2017a	R2017b	R2018a	R2018b			
MATLAB Production Server R2019a				R2016b	R2017a	R2017b	R2018a	R2018b	R2019a		

MATLAB Production Server	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime	MATLAB Runtime
MATLAB Production Server R2019b					R2017a	R2017b	R2018a	R2018b	R2019a	R2019b	
MATLAB Production Server R2020a						R2017b	R2018a	R2018b	R2019a	R2019b	R2020a

See Also

`mps-setup`

More About

- “Specify the Default MATLAB Runtime for New Server Instances” on page 3-4
- “Support Multiple MATLAB Runtime Versions”

Set Up

Create a Server

Before you can deploy your MATLAB code with MATLAB Production Server, you must create a server to host your deployable archive. A server instance is one unique configuration of the MATLAB Production Server product. Each configuration has its own parameter settings file (`main_config`), as well as its own set of diagnostic files.

Prerequisites

Before creating a server, ensure you have:

- installed MATLAB Production Server on your machine. For more information, see “Install MATLAB Production Server” on page 2-2.
- added the `script` folder to your system `PATH` environment variable. Doing so enables you to run server commands such as `mps-new` from any folder on your system.

You can run server commands from the *`$MPS_INSTALL`*`\script` folder, where *`$MPS_INSTALL`* is the location where MATLAB Production Server is installed. For example, on Windows, the default location is `C:\Program Files\MATLAB\MATLAB Production Server\`*`ver`*`\script`. *`ver`* is the version of MATLAB Production Server.

Procedure

To create a server configuration or instance from the command-line in an on-premise installation, enter the `mps-new` command from the system prompt. Specify the name of the server that you want to create as an argument to the `mps-new` command.

```
mps-new [path/]server_name [-v]
```

- *`path`* — path to the server instance and configuration that you want to create for use with the MATLAB Production Server product.

  If you are creating a server instance in the current folder, you do not need to specify a full path. Only specify the server name.
- *`server_name`* — name of the server instance and configuration that you want to create.
- `-v` — enable verbose output to get the information and status about each folder created in the server configuration.

For example, to create a server instance with the name `prod_server_1` located in `C:\tmp` and using the verbose mode, run the following on your system command prompt.

```
C:\tmp>mps-new prod_server_1 -v
```

The command generates the following output.

```
prod_server_1/.mps_version...ok
prod_server_1/config/main_config...ok
prod_server_1/auto_deploy...ok
prod_server_1/x509...ok
prod_server_1/endpoint...ok
prod_server_1/log...ok
prod_server_1/old_logs...ok
prod_server_1/.mps_socket...ok
```

```
prod_server_1/pid...ok
The UUID of the newly created instance is 4876f876-56a6-40ef-a4e3-96a69b39cb49
```

For more information on the folders created in a server configuration, see “Server Diagnostic Tools”.

See Also

`mps-new` | `mps-service`

More About

- “Install a Server Instance as a Windows Service”
- “Start a Server Instance” on page 3-5

Specify the Default MATLAB Runtime for New Server Instances

Each server instance that you create with MATLAB Production Server has its own configuration file that defines various server management criteria. Use the `mps-setup` command to set the default MATLAB Runtime for all on-premise server instances that you create. The `mps-setup` command line wizard searches your machine for installed MATLAB Runtime instances and sets the default path to the MATLAB Runtime for all server instances.

If you do not have MATLAB Runtime installed on your machine, you must install it first. For more information, see “Supported MATLAB Runtime Versions” on page 2-3.

To set the default MATLAB Runtime:

1. Open a system command prompt with `administrator` privileges.
2. From the command prompt, navigate to the MATLAB Production Server `script` folder and run `mps-setup`.

   Alternatively, add the `script` folder to your system `PATH` environment variable to run `mps-setup` from any folder on your system. The `script` folder is located at *`$MPS_INSTALL`*`\script`, where *`$MPS_INSTALL`* is the location in which MATLAB Production Server is installed. For example, on Windows®, the default location for the `script` folder is `C:\Program Files\MATLAB\MATLAB Production Server\`*`ver`*`\script\mps-setup`, where *`ver`* is the version of MATLAB Production Server.
3. Follow the instructions in the command line wizard.

   The wizard searches your system for installed MATLAB Runtime instances and displays them.
4. Enter `y` to confirm or `n` to specify a default MATLAB Runtime for all server instances.

   If `mps-setup` cannot locate an installed MATLAB Runtime on your system, the wizard prompts you to enter a path name to a valid instance.

Run mps-setup in Non-Interactive Mode for Silent Install

You can also run `mps-setup` without interactive command input for silent installations. To do so, specify the path name of the MATLAB Runtime as a command line argument.

For example, on Windows, run the following at the system command prompt.

```
C:\>mps-setup "C:\Program Files\MATLAB\MATLAB Runtime\mcrver"
```

`mcrver` is the version of the MATLAB Runtime to use.

See Also

mcr-root | `mps-setup`

More About

- “Specify the MATLAB Runtime for a Server Instance”
- “Support Multiple MATLAB Runtime Versions”

Start a Server Instance

In this section...

Follow the procedure below to start a server instance in an on-premise installation of MATLAB Production Server.

Prerequisites

Before attempting to start a server, verify that you have:

- Installed the MATLAB Runtime on page 2-3
- Created a server instance on page 3-2
- Specified the default MATLAB Runtime for the instance on page 3-4

Procedure

To start a server instance, complete the following steps:

1. Open a system command prompt.
2. Enter the `mps-start` command:

   ```
   mps-start [-C path/]server_name [-f]
   ```

   where:

   - `-C` *`path/`* — Path to the server instance you want to create. *`path`* should end with the server name.
   - *`server_name`* — Name of the server instance you want to start or stop.
   - `-f` — Forces command to succeed, regardless or whether the server is already started or stopped.

   Note If needed, use the `mps-status` command to verify the server is running.

See Also

mps-new | mps-service

More About

- “Install a Server Instance as a Windows Service”
- “Share the Deployable Archive” on page 5-8

Verify Server Status

In this section...

Use the `mps-status` command to verify the status of a server in an on-premise MATLAB Production Server installation.

Procedure

1. Open a system command prompt.
2. Enter the following command:

   ```
   mps-status [-C path/]server_name
   ```

   where:

   - `-C path/` — Path to the server instance. *`path`* should end with the name of the server to be queried for status.
   - *`server_name`* — Name of the server to be queried for status.

Example

To verify the status of a server instance `prod_server_1` located at `\tmp\prod_server_1`, type at the system command prompt

```
mps-status -C \tmp\prod_server_1
```

Output:

- If `prod_server_1` is running and operating with a valid license.

  ```
  \tmp\prod_server_1 STARTED
  License checked out
  ```

- If `prod_server_1` is unable to check out valid license.

  ```
  \tmp\prod_server_1 STARTED
  WARNING: lost connection to license server -
  request processing will be disabled at 2019-Jun-27
  15:40:31.002137 Eastern Daylight Time unless
  connection to license server is restored.
  ```

  or

  ```
  \tmp\prod_server_1 STARTED
  ERROR: lost connection to license server -
  request processing disabled.
  ```

To verify whether the server has started or stopped after issuing `mps-restart` and `mps-stop` commands, use `mps-status`.

License Server Status Information

In addition to the status of the server, `mps-status` also displays the status of the license server associated with the server you are querying.

License Server Status Message	Message Description
`License checked out`	The server is operating with a valid license. The server is communicating with the License Manager, and the required number of license keys are checked out.
`WARNING: lost connection to license server - request processing will be disabled at` *`time`* `unless connection to license server is restored`	The server has lost communication with the License Manager, but the server is still fully operational and will remain operational until the specified *`time`*. At *`time`*, if connectivity to the license server has not been restored, request processing will be disabled until licensing is reestablished.
`ERROR: lost connection to license server - request processing disabled`	The server has lost communication with the License Manager for a period of time exceeding the grace period. Request processing has been suspended, but the server is actively attempting to reestablish communication with the License Manager. Request processing resumes if the sever is able to reestablish communication with the License Manager.

See Also

`mps-restart` | `mps-stop`

More About

- “Health Check”

Licensing

Manage Licenses for MATLAB Production Server

Complete instructions for installing License Manager can be found in the *MATLAB Installation Guide*.

In addition to following instructions in the License Center to obtain and activate your license, do the following in order to set up and manage licensing for MATLAB Production Server:

Specify or Verify License Server Options in Server Configuration File

Specify or verify values for License Server options in the server configuration file (`main_config`). You create a server by using the `mps-new` command.

Edit the configuration file for the server. Open the file *`server_name`*`/config/main_config` and specify or verify parameter values for the following options. See the comments in the server configuration file for complete instructions and default values.

- `license` — Configuration option to specify the license servers and/or the license files. You can specify multiple license servers including port numbers (*`port_number`*`@`*`license_server_name`*), as well as license files, with one entry in `main_config`. List where you want the product to search, in order of precedence, using semi-colons (;) as separators on Windows or colons (:) as separators on Linux.

  For example, on a Linux system, you specify this value for `license`:

  ```
  --license 27000@hostA:/opt/license/license.dat:27001@hostB:./license.dat
  ```

  The system searches these resources in this order:

  1. `27000@hostA:` (`hostA` configured on port `27000`)
  2. `/opt/license/license.dat` (local license data file)
  3. `27001@hostB:` (`hostB` configured on port `27001`)
  4. `./license.dat` (local license data file)
- `license-grace-period` — The maximum length of time MATLAB Production Server responds to HTTP requests, after license server heartbeat has been lost. See the network license manager documentation for more on heartbeats and related license terminology.
- `license-poll-interval` — The interval of time that must pass, after license server heartbeat has been lost and MATLAB Production Server stops responding to HTTP requests, before license server is polled, to verify and checkout a valid license. Polling occurs at the interval specified by `license-poll-interval` until license has been successfully checked-out. See the network license manager documentation for more on heartbeats and related license terminology.

Verify Status of License Server using mps-status

When you enter an `mps-status` command, the status of the server *and* the associated license is returned. The command is available only in an on-premise installation of MATLAB Production Server.

For detailed descriptions of these status messages, see “License Server Status Information” on page 3-7.

Forcing a License Checkout Using mps-license-reset

Use the `mps-license-reset` command to force the server to checkout a license in an on-premise MATLAB Production Server installation. You can use this command at any time, if you do not want to wait for MATLAB Production Server to verify and checkout a license at an interval established by a server configuration option such as `license-grace-period` or `license-poll-interval`.

Deploying an Application

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)

a = a1 + a2;
```

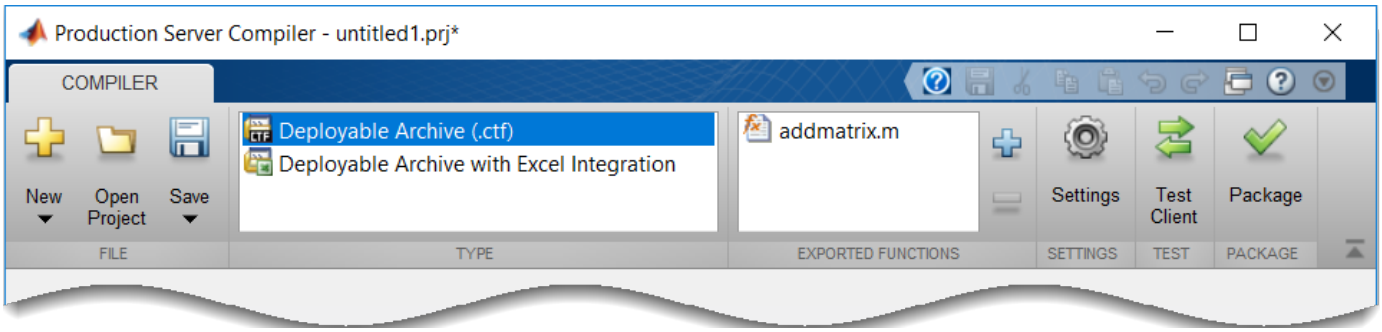
At the MATLAB command prompt, enter `addmatrix([1 4 7; 2 5 8; 3 6 9], [1 4 7; 2 5 8; 3 6 9])`.

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 **Production Server Compiler** 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**.

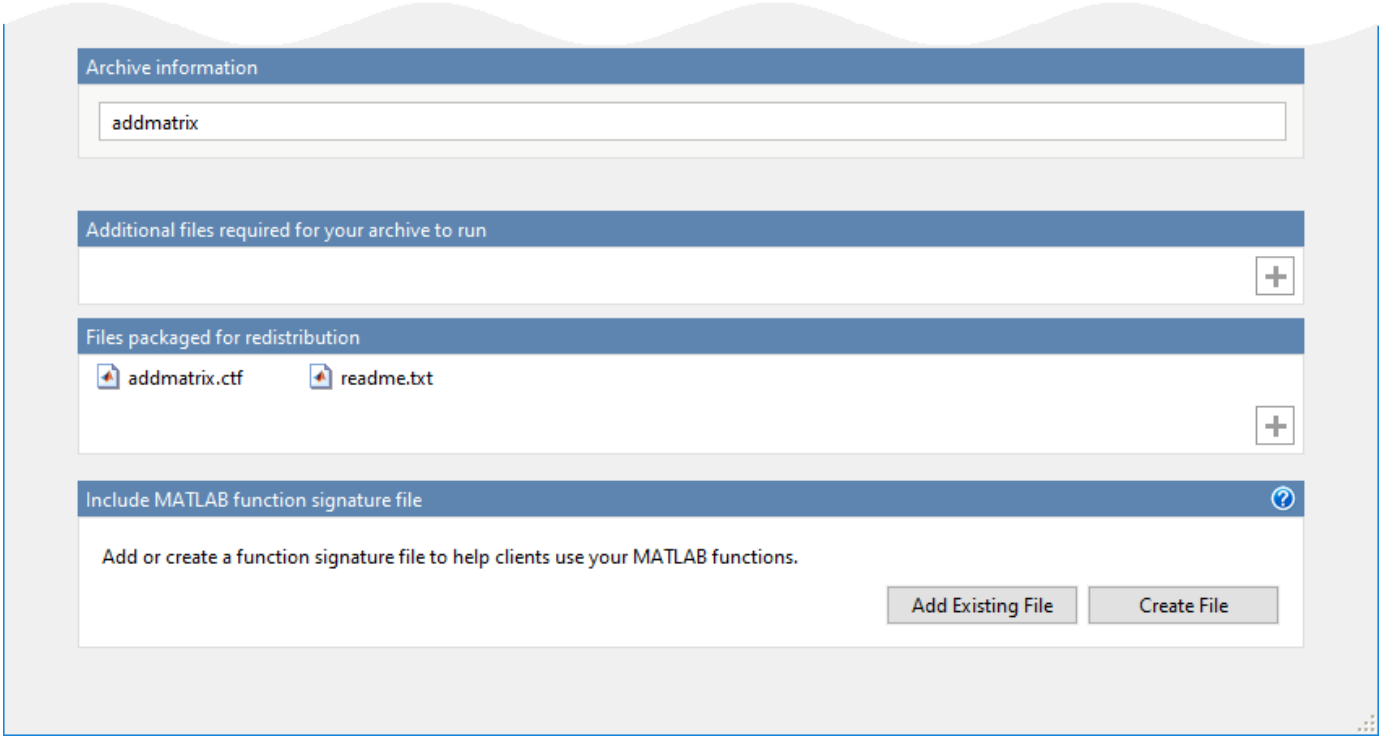
The function `addmatrix.m` is added to the list of main files.

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`

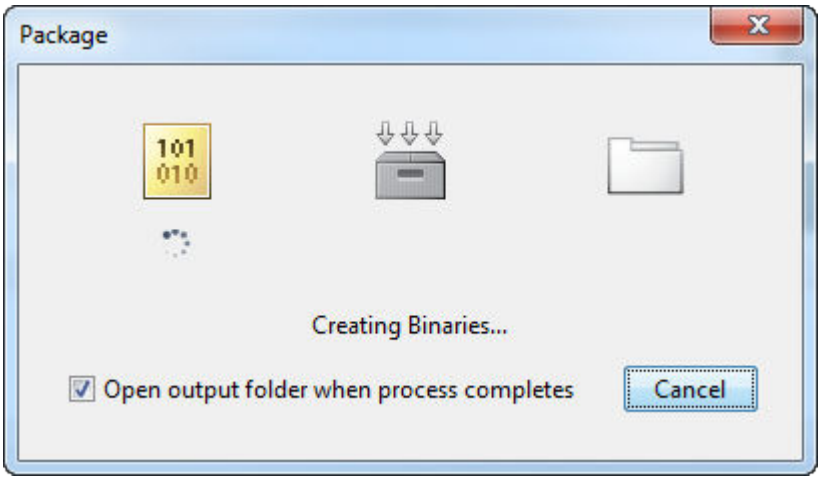
  See “Specify Files to Install with Application” (MATLAB Compiler SDK).
- **Include MATLAB function signature file** — Add or create a function signature file to help clients use your MATLAB functions. See “MATLAB Function Signatures in JSON”.



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
- “Share the Deployable Archive” on page 5-8
- “MATLAB Function Signatures in JSON”

Install and Start MATLAB Production Server

In this section...

This example shows how to install, configure, and start an instance of MATLAB Production Server in an on-premise installation. The example uses the command-line for server creation, configuration and startup.

1. “Install MATLAB Production Server” on page 5-5
2. “Install Network License Manager” on page 5-5
3. “Install MATLAB Runtime” on page 5-6
4. “Specify the Default MATLAB Runtime” on page 5-6
5. “Create a Server Instance” on page 5-6
6. “Configure a Server Instance” on page 5-6
7. “Start a Server Instance” on page 5-7

Install MATLAB Production Server

1 Download the installer from MathWorks Downloads.
2 Run the installer.
3 Select MATLAB Production Server from the product list.
4 When the installer asks where to install MATLAB Production Server, enter the name of an empty folder.
5 Once the installation is complete, add the *`$MPS_INSTALL`*`\script` folder to your system `PATH` environment variable. *`$MPS_INSTALL`*`\script` represents your MATLAB Production Server installation folder.

For more information about installation options, see “Install MATLAB Production Server” on page 2-2.

Install Network License Manager

You must add your MATLAB Production Server license file to the license manager.

1 Download the installer from MathWorks Downloads, if you do not already have it.
2 Run the installer.
3 When the installer prompts you to sign in, click **Advanced Options > I want to install network license manager**.

If your organization already has a network license manager installed, simply add the MATLAB Production Server license file to it.

For more information about installation options, see “Install Network License Manager” on page 2-2.

Install MATLAB Runtime

MATLAB Production Server requires the MATLAB Runtime. You must install the MATLAB Runtime, if you do not have it installed on your system.

1. Download the MATLAB Runtime installer from https://www.mathworks.com/products/compiler/mcr.
2. Run the MATLAB Runtime installer.

For more information, see “Supported MATLAB Runtime Versions” on page 2-3.

Specify the Default MATLAB Runtime

Use the `mps-setup` command to set the default MATLAB Runtime for all the server instances that you create.

To set the default MATLAB Runtime:

1. Open a system command prompt with `administrator` privileges.
2. Run `mps-setup`.

For more information, see “Specify the Default MATLAB Runtime for New Server Instances” on page 3-4.

Create a Server Instance

To create a server configuration or instance, enter the `mps-new` command from the system prompt. Specify the name of the server that you want to create as an argument to the `mps-new` command.

For example, to create a server instance with the name `prod_server_1` located in `C:\tmp` and ro use the verbose mode, run the following on your system command prompt.

```
C:\tmp>mps-new prod_server_1 -v
```

For more information, see “Create a Server” on page 3-2.

Configure a Server Instance

After you create a new server instance, you must configure it. At a minimum, specify the following properties in the `main_config` server configuration file.

- license — Specify the host and port of the license server, typically `27000@license-server-host`.
- mcr-root — Specify the path to the versions of MATLAB Runtime to use. You can set this property to use multiple versions of MATLAB Runtime. For more information, see “Support Multiple MATLAB Runtime Versions”.

For more information on editing `main_config`, see “Edit the Configuration File”.

Start a Server Instance

To start the server instance that you created, enter the `mps-start` command from the system prompt. Specify the name of the server that you want to start as an argument to the `mps-start` command.

For example, to start a server instance with the name `prod_server_1` located in `C:\tmp`, run the following on your system command prompt.

```
C:\tmp>mps-start -C prod_server_1
```

For more information, see “Start a Server Instance”.

See Also

`mps-restart` | `mps-setup` | `mps-start` | `mps-status`

More About

- “Verify Status of License Server using mps-status”
- “Enable HTTPS”

Share the Deployable Archive

After you create a deployable archive, share it with clients of MATLAB Production Server by copying it to your server for hosting. In order for clients to access the deployable archive, you must have a server created and running.

The deployable archive has the name `project_name.ctf`. If you use the **Production Server Compiler** app to create a deployable archive, the archive is available in the `for_redistribution` folder of the deployment project. If you use the `mcc` command to create a deployable archive, you can specify an output folder to create the deployable archive.

For an on-premises server instance, copy the deployable archive into the `auto_deploy` folder of your server instance. You can add a deployable archive into the `auto_deploy` folder of a running server — the server monitors this folder dynamically and processes the deployable archives that are added to the `auto_deploy` folder.

For a server deployment on Azure®, use the dashboard to upload and share applications. For more information, see “Upload MATLAB Application” and “Upload MATLAB Applications”.

See Also

Production Server Compiler | `mcc` | `productionServerCompiler`

More About

- “Create a Deployable Archive with Production Server Compiler App” on page 5-2
- “Package Deployable Archives from Command Line”
- “Azure Deployment for MATLAB Production Server (BYOL)”
- “Azure Deployment for MATLAB Production Server (PAYG)”

Create a Java Client Using the MWHttpClient Class

This example shows how to write a MATLAB Production Server client using the Java client API. In your Java code, you will:

- Define a Java interface that represents the MATLAB function.
- Instantiate a proxy object to communicate with the server.
- Call the deployed function in your Java code.

To create a Java MATLAB Production Server client application:

1. Create a new file called `MPSClientExample.java`.
2. Using a text editor, open `MPSClientExample.java`.
3. Add the following import statements to the file:

```
import java.net.URL;
import java.io.IOException;
import com.mathworks.mps.client.MWClient;
import com.mathworks.mps.client.MWHttpClient;
import com.mathworks.mps.client.MATLABException;
```

4. Add a Java interface that represents the deployed MATLAB function.

   The interface for the `addmatrix` function

```
function a = addmatrix(a1, a2)

a = a1 + a2;
```

   looks like this:

```
interface MATLABAddMatrix {
       double[][] addmatrix(double[][] a1, double[][] a2)
         throws MATLABException, IOException;
   }
```

   When creating the interface, note the following:

   - You can give the interface any valid Java name.
   - You must give the method defined by this interface the same name as the deployed MATLAB function.
   - The Java method must support the same inputs and outputs supported by the MATLAB function, in both type and number. For more information about data type conversions and how to handle more complex MATLAB function signatures, see “Java Client Programming”.
   - The Java method must handle MATLAB exceptions and I/O exceptions.

5. Add the following class definition:

```
public class MPSClientExample
{
}
```

   This class now has a single main method that calls the generated class.

6. Add the `main()` method to the application.

```
public static void main(String[] args)
{
}
```

7 Add the following code to the top of the `main()` method:

```
double[][] a1={{1,2,3},{3,2,1}};
double[][] a2={{4,5,6},{6,5,4}};
```

These statements initialize the variables used by the application.

8 Instantiate a client object using the `MWHttpClient` constructor.

```
MWClient client = new MWHttpClient();
```

This class establishes an HTTP connection between the application and the server instance.

9 Call the client object's `createProxy` method to create a dynamic proxy.

You must specify the URL of the deployable archive and the name of your interface `class` as arguments:

```
MATLABAddMatrix m = client.createProxy(new URL("http://localhost:9910/addmatrix"),
                                        MATLABAddMatrix.class);
```

The URL value (`"http://localhost:9910/addmatrix"`) used to create the proxy contains three parts:

- the server address (`localhost`).
- the port number (`9910`).
- the archive name (`addmatrix`)

For more information about the `createProxy` method, see the Javadoc included in the *`$MPS_INSTALL`*`/client` folder, where *`$MPS_INSTALL`* is the name of your MATLAB Production Server installation folder.

10 Call the deployed MATLAB function in your Java application by calling the public method of the interface.

```
  double[][] result = m.addmatrix(a1,a2);
```

11 Call the client object's `close()` method to free system resources.

```
client.close();
```

12 Save the Java file.

The completed Java file should resemble the following:

```
import java.net.URL;
import java.io.IOException;
import com.mathworks.mps.client.MWClient;
import com.mathworks.mps.client.MWHttpClient;
import com.mathworks.mps.client.MATLABException;

interface MATLABAddMatrix
  {
    double[][] addmatrix(double[][] a1, double[][] a2)
         throws MATLABException, IOException;
  }

public class MPSClientExample {

    public static void main(String[] args){

        double[][] a1={{1,2,3},{3,2,1}};
        double[][] a2={{4,5,6},{6,5,4}};

        MWClient client = new MWHttpClient();
```

```
        try{
            MATLABAddMatrix m = client.createProxy(new URL("http://localhost:9910/addmatrix"),
                                                   MATLABAddMatrix.class);
            double[][] result = m.addmatrix(a1,a2);

            // Print the resulting matrix
            printResult(result);

        }catch(MATLABException ex){

            // This exception represents errors in MATLAB
               System.out.println(ex);
        }catch(IOException ex){

            // This exception represents network issues.
               System.out.println(ex);
        }finally{

            client.close();
        }
    }

    private static void printResult(double[][] result){
        for(double[] row : result){
            for(double element : row){
                System.out.print(element + " ");
            }
            System.out.println();
        }
    }
}
```

13 Compile the Java application, using the `javac` command or use the build capability of your Java IDE.

For example, enter the following:

```
javac -classpath "MPS_INSTALL_ROOT\client\java\mps_client.jar" MPSClientExample.java
```

14 Run the application using the `java` command or your IDE.

For example, enter the following:

```
java -classpath .;"MPS_INSTALL_ROOT\client\java\mps_client.jar" MPSClientExample
```

The application returns the following at the console:

```
5.0 7.0 9.0
9.0 7.0 5.0
```

See Also

More About

- “Bond Pricing Tool for Java Client”

Create a C# Client Using MWHttpClient

This example shows how to write a C# application to call a MATLAB function deployed to MATLAB Production Server. The C# application uses the MATLAB Production Server .NET client library.

A .NET application programmer typically performs this task. The tutorial assumes that you have Microsoft® Visual Studio® and .NET installed on your computer.

Create Microsoft Visual Studio Project

1. Open Microsoft Visual Studio.
2. Click **File > New > Project**.
3. In the New Project dialog box, select the template you want to use. For example, if you want to create a C# console application in Visual Studio 2017, select **Visual C# > Windows Desktop** in the left navigation pane, then select the **Console App (.Net Framework)**.
4. Type the name of the project in the **Name** field (for example, `Magic`).
5. Click **OK**. Your `Magic` source shell is created, typically named `Program.cs`, by default.

Create Reference to Client Runtime Library

Create a reference in your `Magic` project to the MATLAB Production Server client runtime library. In Microsoft Visual Studio, perform the following steps:

1. In the **Solution Explorer** pane within Microsoft Visual Studio (usually on the right side), right-click your `Magic` project, select **Add > Browse**.
2. Browse to the MATLAB Production Server .NET client runtime library location.

   In an on-premises MATLAB Production Server installation, the library is located in *`$MPS_INSTALL`*`\client\dotnet`, where *`$MPS_INSTALL`* is the location in which MATLAB Production Server is installed. Select the `MathWorks.MATLAB.ProductionServer.Client.dll` file.

   The client library is also available for download at `https://www.mathworks.com/products/matlab-production-server/client-libraries.html`.
3. Click **OK**. Your Microsoft Visual Studio project now references the `MathWorks.MATLAB.ProductionServer.Client.dll`.

Design .NET Interface in C#

In this example, you invoke a MATLAB function `mymagic.m` hosted by the server from a .NET client through a .NET interface. The `mymagic` function uses the `magic` function to create a magic square. The function `mymagic` takes a single `int` input and returns a magic square as a 2-D `double` array.

```
function m = mymagic(in)
    m = magic(in);
```

Design a C# interface `Magic` to match the MATLAB function `mymagic.m`. In your C# client program, use this interface to specify the type of the proxy object reference in the `CreateProxy` method.

- The .NET interface has the same number of inputs and outputs as the MATLAB function.
- Since you are deploying one MATLAB function on the server, you define one corresponding .NET method in your C# code.

- Both the MATLAB function and the .NET interface process the same data types—input type `int` and output type 2-D `double`.

```
public interface Magic
       {
         double[,] mymagic(int in1);
       }
```

For information on creating and deploying an archive to the server, see “Create a Deployable Archive for MATLAB Production Server” and “Share the Deployable Archive” on page 5-8.

Write, Build, and Run .NET Application

1. Open the Microsoft Visual Studio project `Magic` that you created earlier.
2. In the `Program.cs` tab, paste in the code below.

```
using System;
using System.Net;
using MathWorks.MATLAB.ProductionServer.Client;

namespace Magic
{
    public class MagicClass
    {

        public interface Magic
        {
            double[,] mymagic(int in1);
        }

        public static void Main(string[] args)
        {
            MWClient client = new MWHttpClient();
            try
            {
                Magic me = client.CreateProxy<Magic>
                          (new Uri("http://localhost:9910/mymagic_deployed"));
                double[,] result1 = me.mymagic(4);
                print(result1);
            }
            catch (MATLABException ex)
            {
                Console.WriteLine("{0} MATLAB exception caught.", ex);
                Console.WriteLine(ex.StackTrace);
            }
            catch (WebException ex)
            {
                Console.WriteLine("{0} Web exception caught.", ex);
                Console.WriteLine(ex.StackTrace);
            }
            finally
            {
                client.Dispose();
            }
            Console.ReadLine();
        }

        public static void print(double[,] x)
        {
            int rank = x.Rank;
            int[] dims = new int[rank];

            for (int i = 0; i < rank; i++)
            {
                dims[i] = x.GetLength(i);
            }

            for (int j = 0; j < dims[0]; j++)
```

```
            {
                for (int k = 0; k < dims[1]; k++)
                {
                    Console.Write(x[j, k]);
                    if (k < (dims[1] - 1))
                    {
                        Console.Write(",");
                    }
                }
                Console.WriteLine();
            }
        }
    }
}
```

The URL value (`"http://localhost:9910/mymagic_deployed"`) used to create the proxy contains three parts.

- the server address (`localhost`).
- the port number (`9910`).
- the archive name (`mymagic_deployed`).

3 Build the application. Click **Build** > **Build Solution**.

4 Run the application. Click **Debug** > **Start Without Debugging**. The program returns the following console output.

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

See Also

More About

- “Create a .NET MATLAB Production Server Client”
- “Configure the Client-Server Connection”
- “Synchronous RESTful Requests Using Protocol Buffers in .NET Client”

Create a C++ Client

This example shows how to write a MATLAB Production Server client using the C client API. The client application calls the `addmatrix` function you compiled in “Package Deployable Archives with Production Server Compiler App” (MATLAB Compiler SDK) and deployed in “Share the Deployable Archive” on page 5-8.

Create a C++ MATLAB Production Server client application:

1. Create a file called `addmatrix_client.cpp`.
2. Using a text editor, open `addmatrix_client.cpp`.
3. Add the following include statements to the file:

```
#include <iostream>
#include <mps/client.h>
```

> **Note** The header files for the MATLAB Production Server C client API are located in the *$MPS_INSTALL*`/client/c/include/mps` folder where *$MPS_INSTALL* is the root folder which MATLAB Production Server is installed.

4. Add the `main()` method to the application.

```
int main ( void )
{
}
```

5. Initialize the client runtime.

```
mpsClientRuntime* mpsruntime = mpsInitializeEx(MPS_CLIENT_1_1);
```

6. Create the client configuration.

```
mpsClientConfig* config;
mpsStatus status = mpsruntime->createConfig(&config);
```

7. Create the client context.

```
mpsClientContext* context;
status = mpsruntime->createContext(&context, config);
```

8. Create the MATLAB data to input to the function.

```
double a1[2][3] = {{1,2,3},{3,2,1}};
double a2[2][3] = {{4,5,6},{6,5,4}};

int numIn=2;
mpsArray** inVal = new mpsArray* [numIn];

inVal[0] = mpsCreateDoubleMatrix(2,3,mpsREAL);
inVal[1] = mpsCreateDoubleMatrix(2,3,mpsREAL);

double* data1 = (double *)( mpsGetData(inVal[0]) );
double* data2 = (double *)( mpsGetData(inVal[1]) );

for(int i=0; i<2; i++)
{
  for(int j=0; j<3; j++)
  {
    mpsIndex subs[] = { i, j };
    mpsIndex id = mpsCalcSingleSubscript(inVal[0], 2, subs);
    data1[id] = a1[i][j];
```

```
    data2[id] = a2[i][j];
  }
}
```

9 Create the MATLAB data to hold the output.

```
int numOut = 1;
mpsArray **outVal = new mpsArray* [numOut];
```

10 Call the deployed MATLAB function.

Specify the following as arguments:

- client context
- URL of the function
- Number of expected outputs
- Pointer to the `mpsArray` holding the outputs
- Number of inputs
- Pointer to the `mpsArray` holding the inputs

```
mpsStatus status = mpsruntime->feval(context,
    "http://localhost:9910/addmatrix/addmatrix",
    numOut, outVal, numIn, (const mpsArray**)inVal);
```

For more information about the `feval` function, see the reference material included in the *$MPS_INSTALL*`/client` folder, where *$MPS_INSTALL* is the name of your MATLAB Production Server installation folder.

11 Verify that the function call was successful using an `if` statement.

```
if (status==MPS_OK)
{
}
```

12 Inside the `if` statement, add code to process the output.

```
double* out = mpsGetPr(outVal[0]);

for (int i=0; i<2; i++)
{
  for (int j=0; j<3; j++)
  {
    mpsIndex subs[] = {i, j};
    mpsIndex id = mpsCalcSingleSubscript(outVal[0], 2, subs);
    std::cout << out[id] << "\t";
  }
  std::cout << std::endl;
}
```

13 Add an `else` clause to the `if` statement to process any errors.

```
else
{
  mpsErrorInfo error;
  mpsruntime->getLastErrorInfo(context, &error);
  std::cout << "Error: " << error.message << std::endl;
  switch(error.type)
  {
    case MPS_HTTP_ERROR_INFO:
      std::cout << "HTTP: " << error.details.http.responseCode << ": "
          << error.details.http.responseMessage << std::endl;
```

```
    case MPS_MATLAB_ERROR_INFO:
      std::cout << "MATLAB: " << error.details.matlab.identifier
          << std::endl;
      std::cout << error.details.matlab.message << std::endl;
    case MPS_GENERIC_ERROR_INFO:
      std::cout << "Generic: " << error.details.general.genericErrorMsg
          << std::endl;
  }

  mpsruntime->destroyLastErrorInfo(&error);
}
```

14 Free the memory used by the inputs.

```
for (int i=0; i<numIn; i++)
  mpsDestroyArray(inVal[i]);
delete[] inVal;
```

15 Free the memory used by the outputs.

```
for (int i=0; i<numOut; i++)
  mpsDestroyArray(outVal[i]);
delete[] outVal;
```

16 Free the memory used by the client runtime.

```
mpsruntime->destroyConfig(config);
mpsruntime->destroyContext(context);
mpsTerminate();
```

17 Save the file.

The completed program should resemble the following:

```
#include <iostream>
#include <mps/client.h>

int main ( void )
{
  mpsClientRuntime* mpsruntime = mpsInitializeEx(MPS_CLIENT_1_1);

  mpsClientConfig* config;
  mpsStatus status = mpsruntime->createConfig(&config);

  mpsClientContext* context;
  status = mpsruntime->createContext(&context, config);

  double a1[2][3] = {{1,2,3},{3,2,1}};
  double a2[2][3] = {{4,5,6},{6,5,4}};

  int numIn=2;
  mpsArray** inVal = new mpsArray* [numIn];
  inVal[0] = mpsCreateDoubleMatrix(2,3,mpsREAL);
  inVal[1] = mpsCreateDoubleMatrix(2,3,mpsREAL);
  double* data1 = (double *)( mpsGetData(inVal[0]) );
  double* data2 = (double *)( mpsGetData(inVal[1]) );
  for(int i=0; i<2; i++)
  {
    for(int j=0; j<3; j++)
    {
      mpsIndex subs[] = { i, j };
      mpsIndex id = mpsCalcSingleSubscript(inVal[0], 2, subs);
      data1[id] = a1[i][j];
      data2[id] = a2[i][j];
    }
  }

  int numOut = 1;
  mpsArray **outVal = new mpsArray* [numOut];

  status = mpsruntime->feval(context,
               "http://localhost:9910/addmatrix/addmatrix",
               numOut, outVal, numIn, (const mpsArray **)inVal);
```

```
  if (status==MPS_OK)
  {
    double* out = mpsGetPr(outVal[0]);

    for (int i=0; i<2; i++)
    {
      for (int j=0; j<3; j++)
      {
        mpsIndex subs[] = {i, j};
        mpsIndex id = mpsCalcSingleSubscript(outVal[0], 2, subs);
        std::cout << out[id] << "\t";
      }
      std::cout << std::endl;
    }
  }
  else
  {
    mpsErrorInfo error;
    mpsruntime->getLastErrorInfo(context, &error);
    std::cout << "Error: " << error.message << std::endl;

    switch(error.type)
    {
    case MPS_HTTP_ERROR_INFO:
      std::cout << "HTTP: "
          << error.details.http.responseCode
          << ": " << error.details.http.responseMessage
          << std::endl;
    case MPS_MATLAB_ERROR_INFO:
      std::cout << "MATLAB: " << error.details.matlab.identifier
          << std::endl;
      std::cout << error.details.matlab.message << std::endl;
    case MPS_GENERIC_ERROR_INFO:
      std::cout << "Generic: "
          << error.details.general.genericErrorMsg
          << std::endl;
    }
    mpsruntime->destroyLastErrorInfo(&error);
  }

  for (int i=0; i<numIn; i++)
    mpsDestroyArray(inVal[i]);
  delete[] inVal;

  for (int i=0; i<numOut; i++)
    mpsDestroyArray(outVal[i]);
  delete[] outVal;

  mpsruntime->destroyConfig(config);
  mpsruntime->destroyContext(context);
  mpsTerminate();
}
```

18 Compile the application.

To compile your client code, the compiler needs access to `client.h`. This header file is stored in `$MPSROOT/client/c/include/mps/`.

To link your application, the linker needs access to the following files stored in `$MPSROOT/client/c/<arch>/lib/`:

Files Required for Linking

Windows	UNIX®/Linux	Mac OS X
`$arch\lib\mpsclient.lib`	`$arch/lib/libprotobuf.so`	`$arch/lib/libprotobuf.dylib`
	`$arch/lib/libcurl.so`	`$arch/lib/libcurl.dylib`
	`$arch/lib/libmwmpsclient.so`	`$arch/lib/libmwmpsclient.dylib`
	`$arch/lib/libmwcpp11compat.so`	

19 Run the application.

To run your application, add the following files stored in `$MPSROOT/client/c/<arch>/lib/` to the application's path:

Files Required for Running

Windows	UNIX/Linux	Mac OS X
`$arch\lib\mpsclient.dll`	`$arch/lib/libprotobuf.so`	`$arch/lib/libprotobuf.dylib`
`$arch\lib\libprotobuf.dll`	`$arch/lib/libcurl.so`	`$arch/lib/libcurl.dylib`
`$arch\lib\libcurl.dll`	`$arch/lib/libmwmpsclient.so`	`$arch/lib/libmwmpsclient.dylib`
	`$arch/lib/libmwcpp11compat.so`	

The client invokes `addmatrix` function on the server instance and returns the following matrix at the console:

```
5.0 7.0 9.0
9.0 7.0 5.0
```

Create a Python Client

This example shows how to write a MATLAB Production Server client using the Python client API. The client application calls the `addmatrix` function you compiled in “Package Deployable Archives with Production Server Compiler App” (MATLAB Compiler SDK) and deployed in “Share the Deployable Archive” on page 5-8.

Create a Python MATLAB Production Server client application:

1. Copy the contents of the *MPS_INSTALL*`\clients\python` folder to your development environment.
2. Open a command line,
3. Change directories into the folder where you copied the MATLAB Production Server Python client.
4. Run the following command.

   ```
   python setup.py install
   ```
5. Start the Python command line interpreter.
6. Enter the following import statements at the Python command prompt.

   ```
   import matlab
   from production_server import client
   ```
7. Open the connection to the MATLAB Production Server instance and initialize the client runtime.

   ```
   client_obj = client.MWHttpClient("http://localhost:9910")
   ```
8. Create the MATLAB data to input to the function.

   ```
   a1 = matlab.double([[1,2,3],[3,2,1]])
   a2 = matlab.double([[4,5,6],[6,5,4]])
   ```
9. Call the deployed MATLAB function.

   You must know the following:

   - Name of the deployed archive
   - Name of the function

   ```
   client_obj.addmatrix.addmatrix(a1,a2)

   matlab.double([[5.0,7.0,9.0],[9.0,7.0,5.0]])
   ```

   The syntax for invoking a function is `client.`*`archiveName`*`.`*`functionName`*`(`*`arg1, arg2, .., `*`[nargout=`*`numOutArgs`*`])`.
10. Close the client connection.

    ```
    client_obj.close()
    ```