MATLAB® Production Server™

Getting Started

R2014**a**
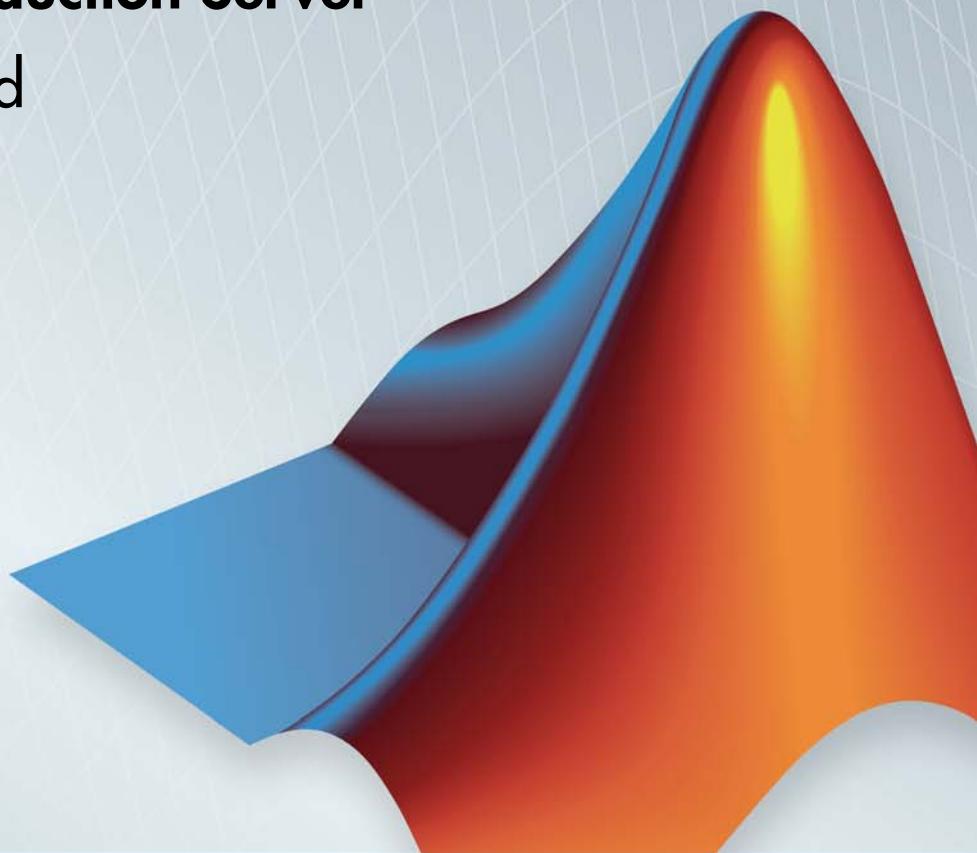
MATLAB®

MathWorks®

How to Contact MathWorks

www.mathworks.com	Web
comp.soft-sys.matlab	Newsgroup
www.mathworks.com/contact_TS.html	Technical Support

suggest@mathworks.com	Product enhancement suggestions
bugs@mathworks.com	Bug reports
doc@mathworks.com	Documentation error reports
service@mathworks.com	Order status, license renewals, passcodes
info@mathworks.com	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

For contact information about worldwide offices, see the MathWorks Web site.

MATLAB® Production Server™ Getting Started Guide

Trademarks

Patents

Revision History

Contents

Licensing

4

Deploying an Application

5

1

Overview

- "MATLAB® Production Server™ Product Description" on page 1-2
- "Roles in Deploying to MATLAB® Production Server™" on page 1-3
- "MATLAB® Production Server™ Workflow" on page 1-5

MATLAB Production Server Product Description

Run MATLAB® programs as part of web, database, and enterprise applications

MATLAB Production Server™ lets you run MATLAB programs within your production systems, enabling you to incorporate numerical analytics in enterprise applications. Web, database, and enterprise applications connect to MATLAB programs running on MATLAB Production Server via a lightweight client library, isolating the MATLAB programs from your production system. You use MATLAB Compiler™ to package programs and deploy them directly to MATLAB Production Server without recoding or creating custom infrastructure to manage them. MATLAB Production Server runs on multiprocessor and multicore servers, providing low-latency processing of many concurrent requests. You can deploy the product on additional computer servers to increase the number of concurrent requests the system can handle and to provide redundancy.

Key Features

- Production deployment of MATLAB programs without recoding or creating custom infrastructure

- Scalable performance and management of packaged MATLAB programs

- Lightweight client library for calling numerical processing programs from enterprise applications

- Common infrastructure across .NET and Java® development environments

- Isolation of MATLAB processes from other system elements

Roles in Deploying to MATLAB Production Server

Deploying MATLAB functionality using MATLAB Production Server is a multistep process that might involve one or more team members. Each step requires fulfilling specific roles, as shown in MATLAB® Production Server™ Deployment Roles on page 1-3 .

MATLAB Production Server Deployment Roles

Role	Knowledge Base	Responsibilities
MATLAB programmer	- MATLAB expert - Little to no software development experience - Little to no IT experience	- Develop functions and implements them in MATLAB. - Create deployable archives that run in MATLAB Production Server instances.
Application developer	- Little to no MATLAB experience - Some knowledge of IT systems - Familiarity with developing applications using a client/server architecture	- Develop applications using one of the MATLAB Production Server client APIs. - Test applications. - Package applications for distribution.

MATLAB Production Server Deployment Roles (Continued)

Role	Knowledge Base	Responsibilities
Server administrator	• Little to no MATLAB experience • Moderate IT experience • Familiarity with IT systems	• Ensure that systems running MATLAB Production Server instances have the required specifications. • Install MATLAB Production Server instances. • Tune MATLAB Production Server instances. • Install compiled MATLAB applications into MATLAB Production Server instances. • Monitor MATLAB Production Server instances.
Application installer	• Little to no MATLAB experience • Moderate IT experience • Familiarity with IT systems	• Ensure that systems using MATLAB Production Server client applications meet the required specifications. • Install any required software on target machines. • Install MATLAB Production Server client applications on target machines.

MATLAB Production Server Workflow

The following figure illustrates the basic workflow to deploy MATLAB code using MATLAB Production Server.
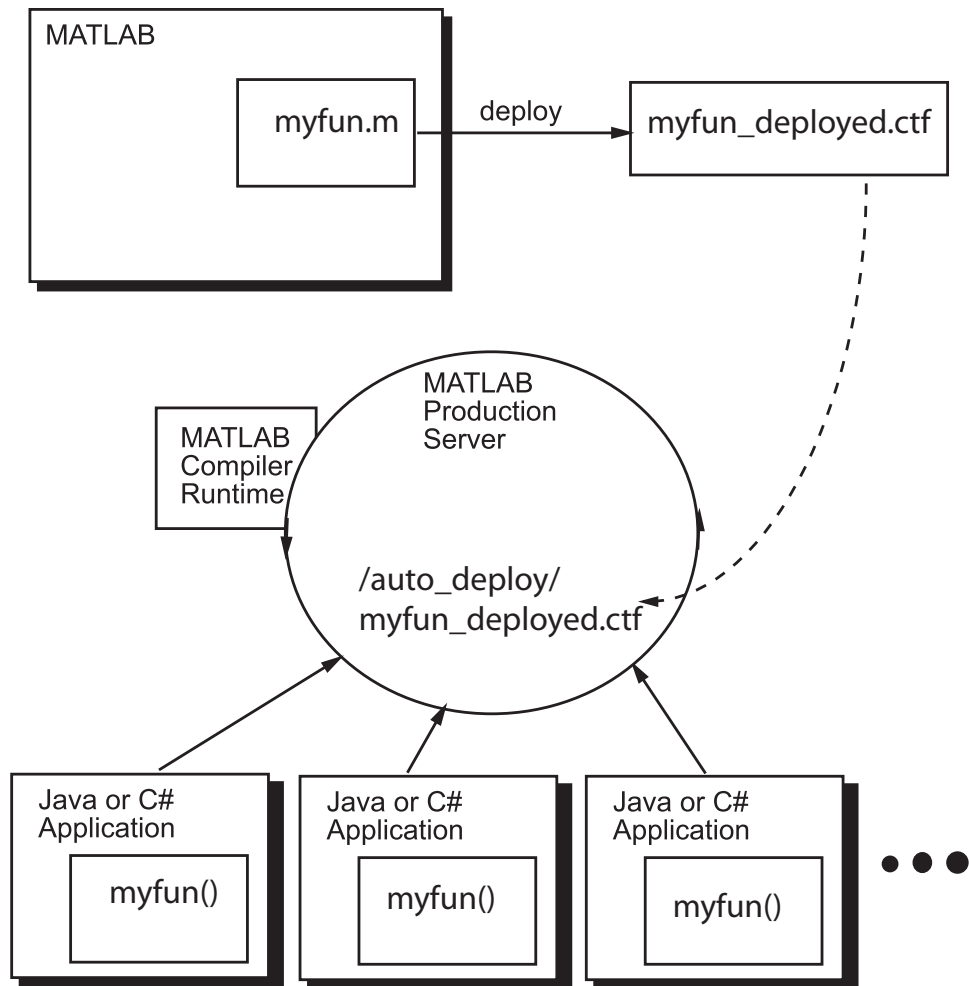
Deploying MATLAB code using MATLAB Production Server is a four-phase process:

1 Create deployable archives.

MATLAB users write MATLAB functions and compile them into deployable archives using MATLAB Compiler.

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 an MCR into a server instance.

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

Application developers use MATLAB Production Server client APIs to write applications that use MATLAB code. MATLAB Production Server client APIs are available for:

- Java
- C#

4 Install client applications on end-user computers.

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

2

Installation

Prerequisites & Compatibility

In this section...
"Windows File System Concerns" on page 2-2
"Architecture Compatibility" on page 2-2

Windows File System Concerns

If you plan to install on Windows, ensure that the system on which you install MATLAB Production Server does not depend on access to files located on a network drive. For stable results in a production environment, servers created with MATLAB Production Server should always have *local* access to the deployable archives that they host.

Architecture Compatibility

Consider if the computers running MATLAB, as well as server instances of MATLAB Production Server that host your code, are 32-bit or 64-bit.

Your operating system and bit architectures must be compatible (or ideally, the same) across machines running MATLAB Production Server and your deployed components.

For additional compatibility considerations, see the MATLAB documentation.

Note You can install a 32-bit image of MATLAB Production Server on a 64-bit version of Windows®.

If you do so, you will receive a message prompting you to run `set MPS_ARCH=win32`.

Install MATLAB Production Server

1 Insert the installation DVD into your computer. If the MathWorks® Installer does not automatically start, run `setup.exe`.

2 Follow the instructions in the Installation Wizard. Select the **Custom** installation option to verify what will be installed. For help completing the wizard, see the *MATLAB Installation Guide*. As you run the installation wizard, note the following:

- If you do not already have the License Manager installed, you must install it. On the Installation Type dialog box, choose to perform a **Custom** installation. Then, on the Product Selection dialog box, select the License Manager for installation. By default, it is not selected for installation.

- If you install the product using the internet, you will be taken to the Licensing Center to complete the licensing process.

Download and Install the MATLAB Compiler Runtime (MCR)

The MATLAB Compiler Runtime (MCR) is a standalone set of shared libraries that enables the execution of compiled MATLAB applications or components on computers that do not have MATLAB installed. MATLAB Production Server requires an MCR instance to execute the deployed MATLAB applications it hosts.

Note Download and install the required version of the MATLAB Compiler Runtime (MCR) from the Web at `http://www.mathworks.com/products/compiler/mcr`.

In order to host a deployable archive created with the Server Archive Compiler, you install a version of the MCR that is compatible with the version of MATLAB you used to create your archive.

For more information about the MCR, including alternate methods of installing it, see "Distributing MATLAB Code Using the MATLAB Compiler Runtime (MCR)" in the MATLAB Compiler section of MathWorks Documentation Center.

Disable Windows Interactive Error Reporting

If the system on which you are running MATLAB Production Server is not monitored frequently, you may want to disable Windows Interactive Error Reporting, using the `DontShowUI` Windows Error Reporting (WER) setting, to avoid processing disruptions.

See WER Settings for Windows Development at `http://msdn.microsoft.com/en-us/library/windows/desktop/bb513638(v=vs.85).aspx` for complete information.

Set Up

- "Create a Server" on page 3-2
- "Start a Server" on page 3-5
- "Verify Server Status" on page 3-7

Create a Server

In this section...

Prerequisites

Before creating a server, ensure you have:

- Installed MATLAB Production Server software.

- 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.

> **Note** You can run server commands from the script folder. The script folder is located at *$MPS_INSTALL*\script, where *$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

Before you can deploy your MATLAB code with MATLAB Production Server, you need to create a server to host your deployable archive.

A server instance is considered to be 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.

To create a server configuration or *instance*, do the following:

1 From the system command prompt, navigate to where you want to create your server instance.

2 Enter the following command from the system prompt:

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

where:

- *path* is the path to the server instance and configuration you want to create for use with the MATLAB Production Server product. When specifying a path, ensure the path ends with the *server_name*.

  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* — is the name of the server instance and configuration you want to create.

- -v — enables verbose output, giving you information and status about each folder created in the server configuration.

Upon successful completion of the command, MATLAB Production Server creates a new server instance.

Run mps-setup to Set Location of MATLAB Compiler Runtime (MCR)

Each server that you create with MATLAB Production Server has its own configuration file that defines various server management criteria.

The mps-setup command line wizard searches for MCR instances and sets the default path to the MATLAB Compiler Runtime (MCR) for all server instances you create with the product.

To run the command line wizard, do the following after first downloading and performing the "Download and Install the MATLAB Compiler Runtime (MCR)" on page 2-4:

1 Ensure you have logged on with administrator privileges.

2 At the system command prompt, run mps-setup from the script folder. Alternately, 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 is: C:\Program Files\MATLAB\MATLAB Production Server*ver*\script\mps-setup.

ver is the version of MATLAB Production Server to use.

3 Follow the instructions in the command line wizard. The wizard will search your system and display installed MCR instances.

4 Enter y to confirm or n to specify a default MCR for all server configurations created with MATLAB Production Server.

If mps-setup cannot locate an installed MCR on your system, you will be prompted 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 run mps-setup, specify the path name of the MCR as a command line argument. For example, on Windows:

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

mcrver is the version of the MCR to use.

Start a Server

In this section...
"Prerequisites" on page 3-5
"Procedure" on page 3-5
"For More Information" on page 3-6

Prerequisites

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

- Installed the MATLAB Compiler Runtime (MCR)

- Created a server

- Customized the server configuration file, main_config with the location of the MCR or Run mps-setup to set the location of the MATLAB Compiler Runtime (MCR)

Procedure

To start a server, complete the following steps:

1 Open a system command prompt.

2 Enter the following 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.

Upon successful completion of the command, the server instance is active.

Note If needed, query the status of the server instance that you started to verify the server is running.

For More Information

For information about...	See...
Downloading and installing the MCR	"Download and Install the MATLAB Compiler Runtime (MCR)" on page 2-4
Solving errors when starting or stopping a server	"Server Troubleshooting"
The mps-start command	mps-start
Stopping the server with the mps-stop command	mps-stop
Verifying status of a server with the mps-status command	mps-status
Product installation	"Install MATLAB® Production Server™" on page 2-3

Verify Server Status

In this section...

Procedure

To verify the status of a server instance, complete the following steps:

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 and configuration you want to create. `path` should end with the server name.

- `server_name` — Name of the server instance and configuration you want to start or stop.

Upon successful completion of the command, the server status displays.

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 verifying.

Possible statuses and their meanings follow:

This License Server Status Message...	Means...
`License checked out`	The server is operating with a valid license. The server is communicating with the License Manager, and the proper 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 actively attempts to reestablish communication with the License Manager until it succeeds, at which time normal request processing resumes. For information about grace periods, see "Specify or Verify License Server Options in Server Configuration File" on page 4-2.

Verify Status of a Server

This example shows how to verify the status of the server instance you started in the previous example.

In this example, you verify the status of prod_server_1, from a location other than the server instance folder (C:\tmp\prod_server_1).

1 Open a system command prompt.

2 To verify prod_server_1 is running, enter this command:

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

If prod_server_1 is running, the following status is displayed:

```
\tmp\prod_server_1 STARTED
license checked out
```

This output confirms prod_server_1 is running and the server is operating with a valid license.

For more information on the STOPPED status, see mps-stop and mps-restart.

For more information about license status messages, see "License Server Status Information" on page 3-7.

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:

  ```
  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 FLEXlm® 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 FLEXlm 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.

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 MATLAB Production Server to checkout a license. You can use this command at any time, providing 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`.

5

Deploying an Application

Create a Deployable Archive for MATLAB Production Server

This example shows how to create a deployable archive for MATLAB Production Server using a MATLAB function. You can then hand the generated archive to a system administrator who will deploy it into MATLAB Production Server.

To create a deployable archive:

1 In MATLAB, examine the MATLAB code that you want to deploy.

   a Open addmatrix.m.

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

a = a1 + a2;
```

   b At the MATLAB command prompt, enter addmatrix(1,2).

     The output appears as follows:

```
ans =

    3
```

2 Open the **Production Server Compiler**.

   a On the toolstrip, select the **Apps** tab.

   b Click the arrow on the far right of the tab to open the apps gallery.

   c Click **Production Server Compiler**.
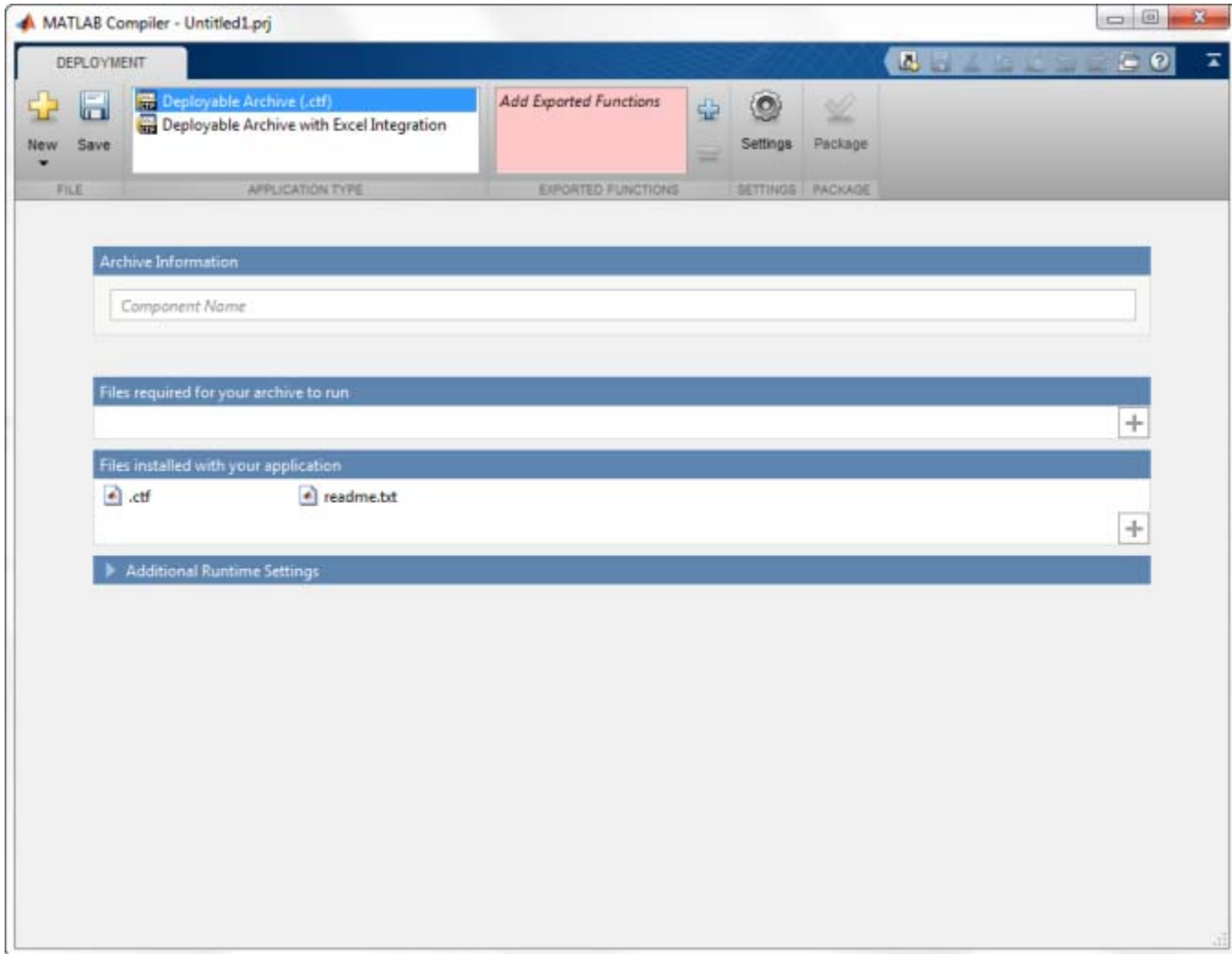
3 In the **Application Type** section of the toolstrip, select **Deployable Archive** from the list.

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

---

4 Specify the MATLAB functions you want to deploy.

    a In the **Exported Functions** section of the toolstrip, click the plus button.

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

---

    b Using the file explorer, locate and select the addmatrix.m file.

       addmatrix.m is located in *matlabroot*\extern\examples\compiler.

    c Click **Open** to select the file and close the file explorer.

       addmatrix.m is added to the field. A minus button will appear below the plus button.

5 In the **Packaging Options** section of the toolstrip, verify that the **Runtime downloaded from web** check box is selected.

Note If the **Packaging Options** section of the toolstrip is collapsed you can expand it by clicking the down arrow.

This option creates an application installer that automatically downloads the MATLAB Compiler Runtime (MCR) and installs it.

6 Explore the main body of the project window.

The project window is divided into the following areas:

- **Application Information** — Editable information about the deployed archive. This information is used by the generated installer to populate the installed application's metadata. See "Customize the Installer".

- **Additional Installer Options** — The default installation path for the generated installer. See "Customize the Installer".

- **Files required for your application** — Additional files required by the archive. These files will be included in the generated archive. See "Manage Required Files in a Compiler Project".

- **Files installed with your application** — Files that are installed with your archive. These files include:

  - readme.txt

  - .ctf file

  See "Specify Files to Install with the Application".

7 Click **Package**.

The Package window opens while the library is being generated.
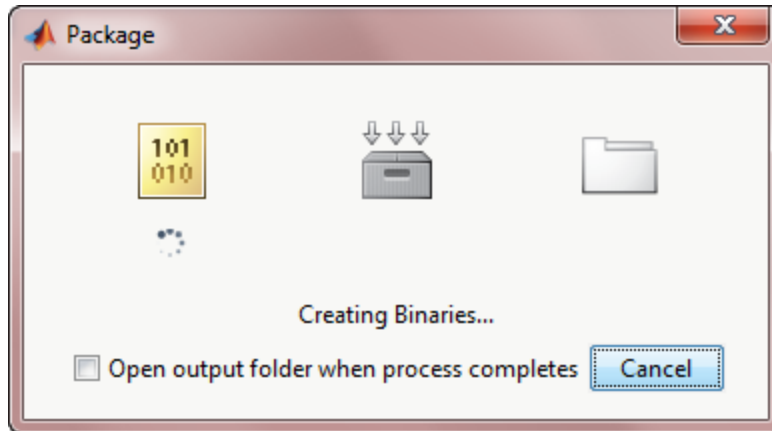
8 Select the **Open output folder when process completes** check box.

When the deployment process is complete, a file explorer opens and displays the generated output.

9 Verify the contents of the generated output:

- for_redistribution — A folder containing the installer to redistribute the archive to the system administrator responsible for the MATLAB Production Server

- for_testing — A folder containing the raw files generated by the compiler

- for_redistribution_files_only — A folder containing only the files needed to redistribute the archive

- PackagingLog.txt — A log file generated by the compiler.

10 Click **Close** on the Package window.

To lean more about MATLAB Production Server see "MATLAB Production Server"

Start a MATLAB Production Server Instance

Overview

This example shows how to install, configure, and start an instance of MATLAB Production Server.

To start a MATLAB Production Server instance:

1 Install MATLAB Production Server.

2 Install MATLAB Compiler Runtime (MCR).

3 Create a server instance.

4 Configure the server instance.

5 Start the server instance.

Install MATLAB Production Server

To install MATLAB Production Server:

1 Run the installer.

2 On the Installation Type dialog box, select **Custom**.

3 Select License Manager for installation in the product list.

4 When asked where to install MATLAB Production Server, enter the name of an empty folder.

You need the path to the installation to complete the tutorial.

5 Add the *$MPS_INSTALL*\script folder to your system PATH environment variable.

$MPS_INSTALL represents your MATLAB Production Server installation folder.

Install MATLAB Compiler Runtime (MCR)

If it is not already installed on your system, you must install the MCR. MATLAB Production Server requires the MCR.

To install an MCR and configure MATLAB Production Server to use it:

1 Download the MCR installer from http://www.mathworks.com/products/compiler/mcr.

2 Run the MCR installer.

For more information, see "Run mps-setup to Set Location of MATLAB Compiler Runtime (MCR)".

Create a Server Instance

To create the server instance:

1 Move to the folder where you want to create your server.

2 Run the mps-new command.

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

3 Verify the output.

```
prod_server_1/.mps_version...ok
prod_server_1/config/main_config...ok
prod_server_1/auto_deploy/...ok
```

```
prod_server_1/log/...ok
prod_server_1/pid/...ok
prod_server_1/old_logs/...ok
prod_server_1/.mps_socket/...ok
prod_server_1/endpoint/...ok
```

For more information on these folders, see "Server Diagnostic Tools".

Configure the Server Instance

After you create a new server instance, you must configure it. The MATLAB Production Server configuration file, main_config, includes many parameters you can use to tune server performance. At a minimum, you must use the file to specify the location of the MCR you want to use with the server.

To configure the server instance's default MCR:

1 From the system command line, run mps-setup.

2 Follow the directions to specify which MCR version MATLAB Production Server uses.

For more information, see "Run mps-setup to Set Location of MATLAB Compiler Runtime (MCR)".

For more information about configuration options, see "Edit the Configuration File".

Start the Server

To start the server:

1 Run the mps-start command.

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

2 Verify the server instance has started using the mps-status command.

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

```
'C:\tmp\prod_server_1' STARTED
```

```
license checked out
```

Share a Deployable Archive on the Server Instance

To make your deployable archive available using MATLAB Production Server, you must copy the deployable archive into the `auto_deploy` folder in 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.

To share the deployable archive created in "Create a Deployable Archive for MATLAB® Production Server™" on page 5-2, copy the deployable archive from the deployment project's `for_redistribution_files_only` folder into the server's `auto_deploy` folder.

Create a Java Application That Calls a Deployed Function

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 addmatrix_client.java.

2 Using a text editor, open addmatrix_client.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/addma
                                        MATLABAddMatrix.class);
```

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 magic square


            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 (on one line):

```
H:\Work>javac -classpath "MPS_INSTALL_ROOT\client\java\mps_client.jar" addmatrix_client.java
```

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

For example, enter the following (on one line):

```
H:\Work>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
```

Create a C# .NET Application That Calls a Deployed Function

This example shows how to call a deployed MATLAB function from a C# application using MATLAB Production Server.

In your C# code, you must:

- Create a Microsoft® Visual Studio® Project.
- Create a Reference to the Client Run-Time Library.
- Design the .NET interface in C#.
- Write, build, and run the C# application.

This task is typically performed by .NET application programmer. This part of the tutorial assumes you have Microsoft Visual Studio and .NET installed on your computer.

Create a Microsoft Visual Studio Project

1 Open Microsoft Visual Studio.

2 Click **File > New > Project**.

3 In the New Project dialog, select the project type and template you want to use. For example, if you want to create a C# Console Application, select **Windows** in the **Visual C#** branch of the **Project Type** pane, and select the **C# Console Application** template from the **Templates** pane.

4 Type the name of the project in the **Name** field (Magic, for example).

5 Click **OK**. Your Magic source shell is created, typically named Program.cs, by default.

Create a Reference to the Client Run-Time Library

Create a reference in your MainApp code to the MATLAB Production Server client run-time library. In Microsoft Visual Studio, perform the following steps:

1 In the Solution Explorer pane within Microsoft Visual Studio (usually on the right side), select the name of your project, Magic, highlighting it.

2 Right-click Magic and select **Add Reference**.

3 In the Add Reference dialog box, select the **Browse** tab. Browse to the MATLAB Production Server client runtime, installed at *$MPS_INSTALL*\client\dotnet. Select MathWorks.MATLAB.ProductionServer.Client.dll.

4 Click **OK**. MathWorks.MATLAB.ProductionServer.Client.dll is now referenced by your Microsoft Visual Studio project.

Design the .NET Interface in C#

In this example, you invoke mymagic.m, hosted by the server, from a .NET client, through a .NET interface.

To match the MATLAB function mymagic.m, design an interface named Magic.

For example, the interface for the mymagic function:

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

might look like this:

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

Note the following:

- The .NET interface has the same number of inputs and outputs as the MATLAB function.

- You are deploying one MATLAB function, therefore you define one corresponding .NET method in your C# code.

- Both MATLAB function and .NET interface process the same types: input type int and the output type two-dimensional double.

- You specify the name of your deployable archive (`magic`, which resides in your `auto_deploy` folder) in your URL, when you call `CreateProxy` (`"http://`*`localhost`*`:9910/magic"`).

Write, Build, and Run the .NET Application

Create a C# interface named `Magic` in Microsoft Visual Studio by doing the following:

1 Open the Microsoft Visual Studio project, `MagicSquare`, that you created earlier.

2 In `Program.cs` tab, paste in the code below.

> **Note** Take care to ensure you reference the precise name of the deployable archive you are hosting on your server, as well as the port number where your server listens for client requests. For example, in the code below, the URL value (`"http://localhost:9910/mymagic_deployed"`) contains both the archive name (`mymagic_deployed`) and the port number (`9910`).

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

namespace Magic
{
  public class MagicClass
  {

  class CustomConfig : MWHttpClientConfig
    {
     public int TimeoutMilliSeconds
       {
        get { return 120000; }
       }
  }

    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("{O} MATLAB exception caught.", ex);
      Console.WriteLine(ex.StackTrace);
    }
    catch (WebException ex)
    {
      Console.WriteLine("{O} 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();
      }
    }
  }
}
```

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
```