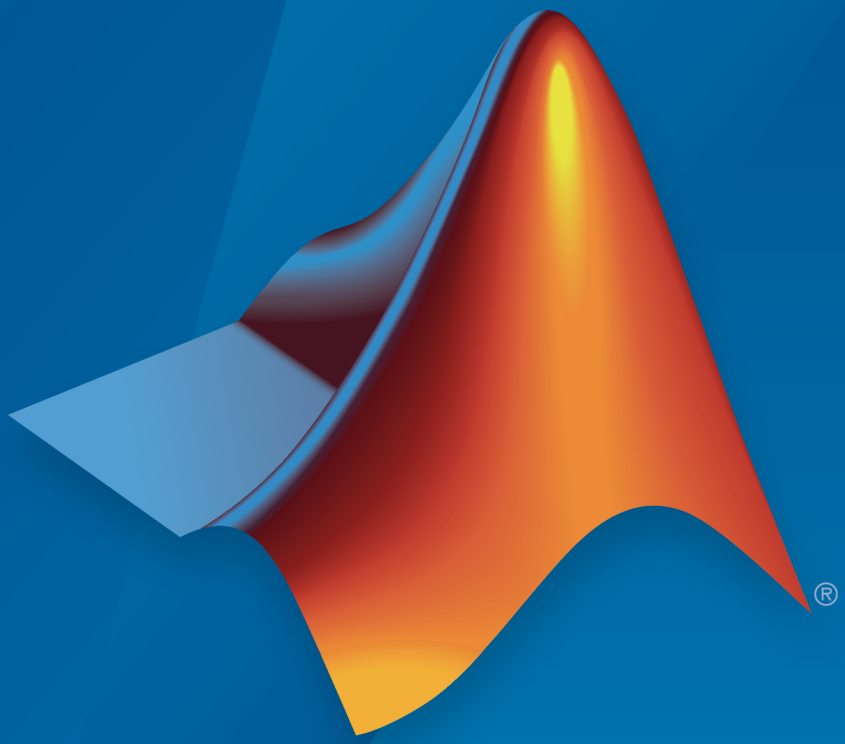
MATLAB® Compiler SDK™

MATLAB® Production Server™ Testing Guide



MATLAB®

R2015a

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

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

Phone: 508-647-7000

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

MATLAB® Compiler SDK™ MATLAB® Production Server™ Testing Guide

Revision History

March 2015	Online only	New for Version 6.0 (Release R2015a)

Contents

1

MATLAB Production Server Integration Testing

Write a Test Client

Integration testing with the MATLAB embedded server instance requires a client that calls the compiled MATLAB functions. The client can be coded using any of the MATLAB Production Server client APIs.

At a minimum, the client must:

1. Instantiate the client runtime.
2. Connect to the embedded server instance using the port specified in the Production Server Compiler app.
3. Call the functions being tested with appropriate data.

For information on writing client code, see:

- "Java Client Programming"
- ".NET Client Programming"
- "C Client Programming"
- "Python Client Programming"

Test Client Data Integration Against MATLAB

This example shows how to test the integration between a Java® client and the `addmatrix` MATLAB function. Using the Production Server Compiler app, you start a testing session and inspect the values passed between the client and the MATLAB function. You also set breakpoints in the MATLAB function and inspect the data using the MATLAB debugger.

1. In MATLAB, enter the function to deploy on MATLAB Production Server.

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

a = a1 + a2;
```

2. In a text editor, paste the following Java client code.

```
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{
```

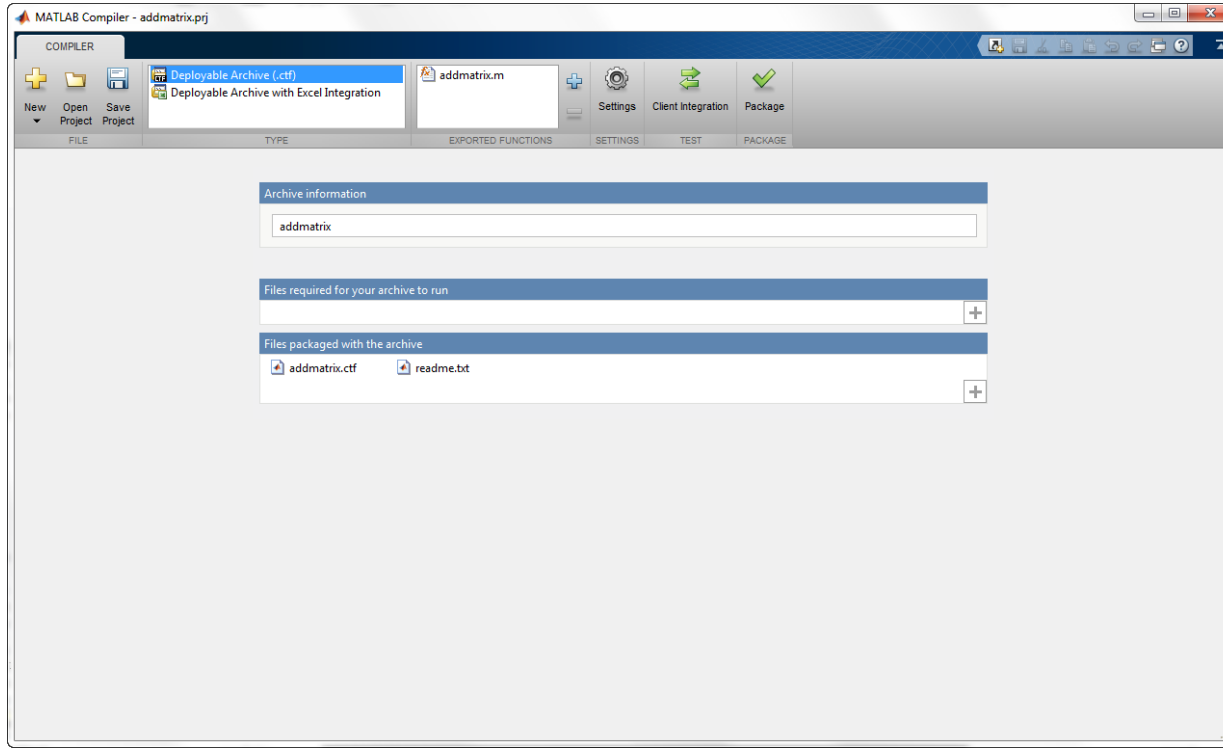
```
            client.close();
        }
    }

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

3. Save the file as `MPSClientExample.java`.
4. At the system command prompt, compile the Java client using the `javac` command.

```
javac -classpath "matlabroot\toolbox\compiler_sdk\mps_client\java\mps_client.jar" MPSClientExample.java
```

5. In MATLAB, open the Production Server Compiler app.

   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**.



6 In the **Type** section of the toolstrip, select **Deployable Archive** from the list.

7 Specify the MATLAB functions to deploy.

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

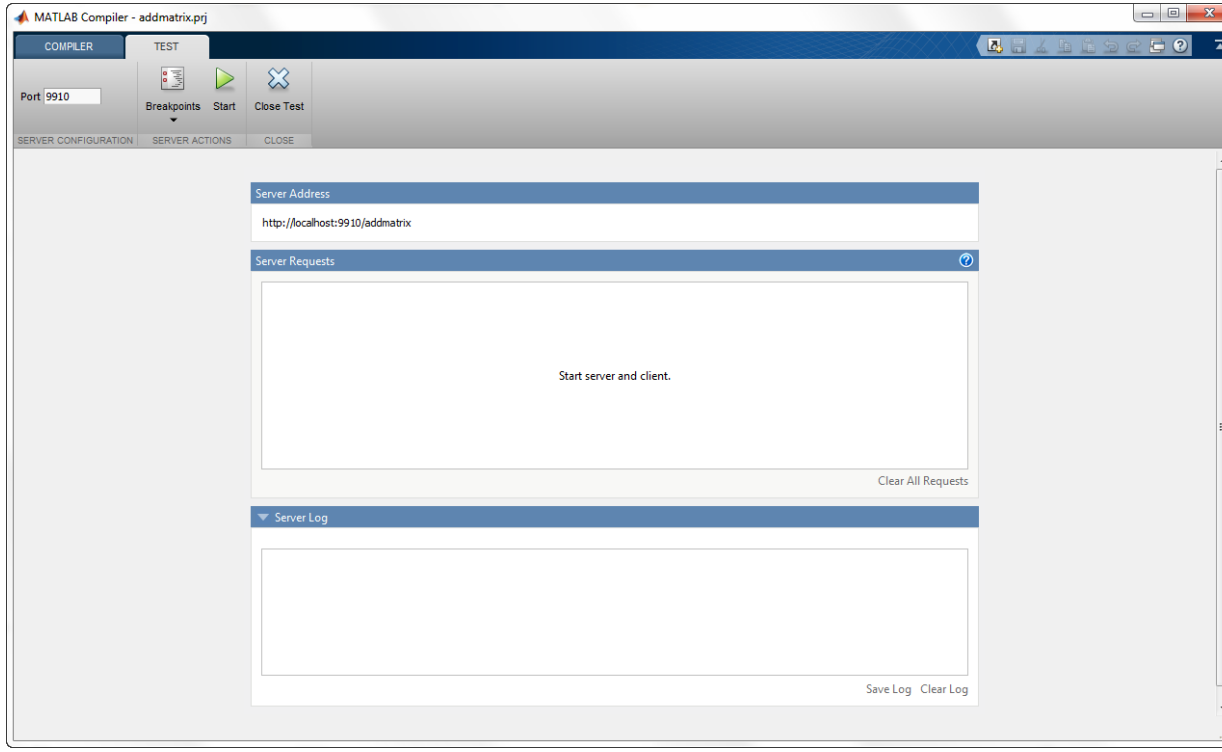
   b Using the file explorer, locate and select the `addmatrix.m` file.

      `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.

8 Click **Client Integration**.



9 Check the value of the **Port** field.

It must be:

- an available port
- the same port number the client is using

For this example, the client expects use the port 9910. If the port is not available, you will need to update the client and recompile it.

10 Click **Start**.

11 At the system command prompt, run the Java client.

```
java -classpath .;"matlabroot\toolbox\compiler_sdk\mps_client\java\mps_client.jar" MPSClientExample
```

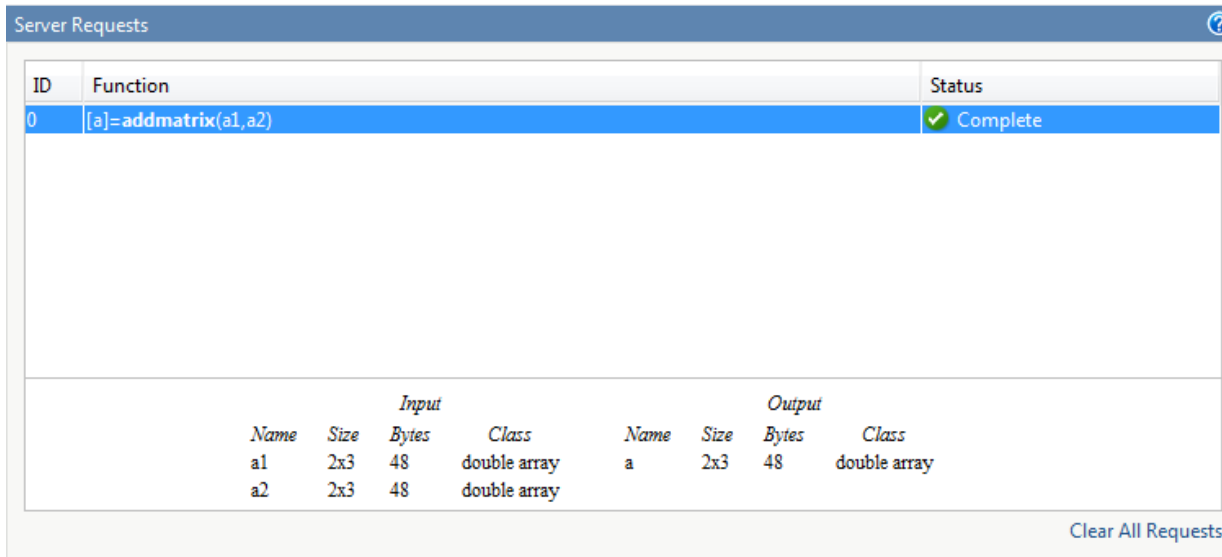
Note: You cannot run the Java client from the MATLAB command prompt.

The application returns the following at the console:

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

The **Server Requests** section of the app shows that the request completed successfully.

12 Click the completed message in the app to see the values exchanged between the client and MATLAB.



13 Click **Breakpoints > Break on MATLAB function entry**.

14 At the system command prompt, run the Java client.

```
java -classpath .;"matlabroot\toolbox\compiler_sdk\mps_client\java\mps_client.jar" MPSClientExample
```

Note: You cannot run the Java client from the MATLAB command prompt.

15 When the MATLAB editor opens, note that a breakpoint is set at the first line in the function and that processing has paused at the breakpoint.

You now can use all of the MATLAB debugging tools to step through your function.

Note: You can create a timeout error in the client if you take a long time stepping through the MATLAB function.

16 Switch to the main MATLAB window.
17 Note that variables `a1` and `a2` are displayed in the MATLAB workspace.
18 In the MATLAB editor, click **Continue**.

The application returns the following at the console:

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

The **Server Requests** section of the app shows that the request completed successfully.
19 Click **Stop** to shutdown the test server.
20 Click **Close Test**.

Related Examples

- "Write a Test Client" on page 1-2
- "Compile Deployable Archives with Production Server Compiler App"

Functions — Alphabetical List

productionServerCompiler

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

Syntax

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

Description

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

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

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

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

Examples

Create a New Production Server Project

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

```
productionServerCompiler
```

Package a Deployable Archive using an Existing Project

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

```
productionServerCompiler -package my_magic
```

Input Arguments

`project_name` — name of the project to be compiled

string

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

Introduced in R2014a

Apps — Alphabetical List

Production Server Compiler

Test and Package MATLAB functions for deployment to MATLAB Production Server

Description

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

Open the Production Server Compiler App

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

Examples

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

More About

- "Deployable Archive Creation"
- "Compile MATLAB Functions"

Parameters

Type

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

Default: Deployable Archive (.ctf)

Settings

Deployable Archive (.ctf)

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

Deployable Archive with Excel Integration

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

Exported Functions

Specify the MATLAB functions to package.

Settings

Specify the output folders for the packaged code.

Default:

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

Archive Information

Specify the a name for the packaged application.

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

Files required for your archive to run

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

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

Files installed with your application

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

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

Programmatic Use

`productionServerCompiler`

Introduced in R2013b