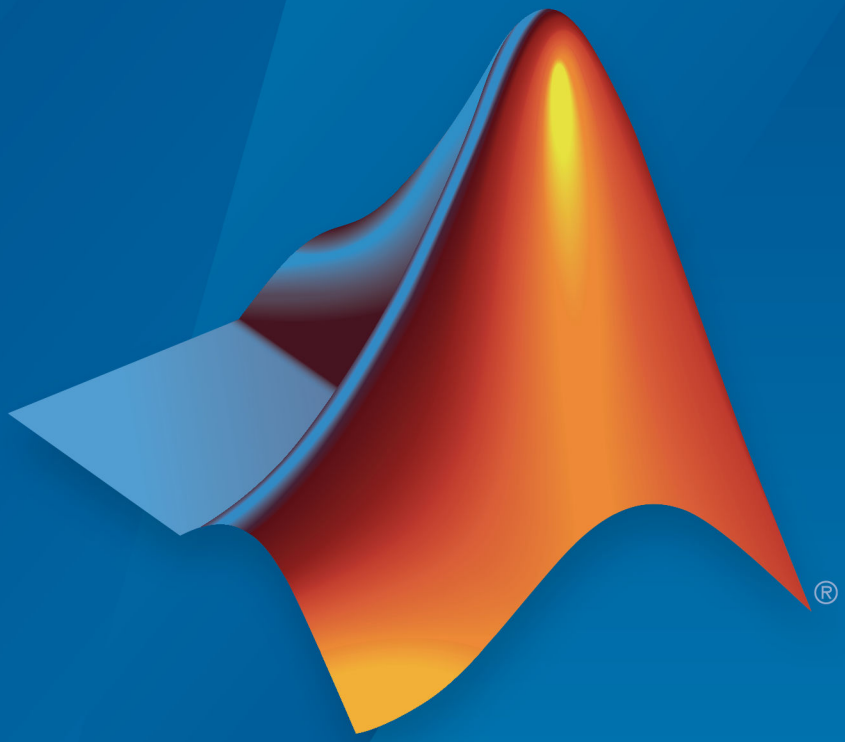
MATLAB® Compiler SDK™

Web Example Guide



MATLAB®

R2019a

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® Compiler SDK™ Web Example Guide

Revision History

March 2008	Online only	New for Version 4.8 (Release R2008a)
October 2008	Online only	Revised for Version 4.9 (Release R2008b)
March 2009	Online only	Revised for Version 4.10 (Release R2009a)
September 2009	Online only	Revised for Version 4.11 (Release R2009b)
March 2010	Online only	Revised for Version 4.13 (Release R2010a)
September 2010	Online only	Revised for Version 4.14 (Release R2010b)
April 2011	Online only	Revised for Version 4.15 (Release R2011a)
September 2011	Online only	Revised for Version 4.16 (Release R2011b)
March 2012	Online only	Revised for Version 4.17 (Release R2012a)
September 2012	Online only	Revised for Version 4.18 (Release R2012b)
September 2013	Online only	Revised for Version 5.0 (Release R2013b)
March 2014	Online only	Revised for Version 5.1 (Release R2014a)
October 2014	Online only	Revised for Version 5.2 (Release R2014b)
March 2015	Online only	Revised for Version 6.0 (Release R2015a)
September 2015	Online only	Revised for Version 6.1 (Release 2015b)
October 2015	Online only	Rereleased for Version 6.0.1 (Release 2015aSP1)
March 2016	Online only	Revised for Version 6.2 (Release 2016a)
September 2016	Online only	Revised for Version 6.3 (Release R2016b)
March 2017	Online only	Revised for Version 6.3.1 (Release R2017a)
September 2017	Online only	Revised for Version 6.4 (Release R2017b)
March 2018	Online only	Revised for Version 6.5 (Release R2018a)
September 2018	Online only	Revised for Version 6.6 (Release R2018b)
March 2019	Online only	Revised for Version 6.6.1 (Release R2019a)

Contents

1 How to Use This Guide

2 Web 101: An Introduction to Web Concepts and Terminology

3 MATLAB Programmer Tasks

4 Middle-Tier Developer Tasks

5 Front-End Web Developer Tasks

6 Server Administrator Tasks

7 End User Tasks

8 End-to-End Developer Tasks

1

How to Use This Guide

Use MATLAB Compiler SDK Web Example Guide

The *MATLAB Compiler SDK Web Example Guide* provides a series of templates for to successfully deploying MATLAB functions on the web.

Use *MATLAB Compiler SDK Web Example Guide* to:

- Learn about the components of a web deployment environment.
- Review an architectural configuration of a typical web deployment implementation and how the components in the configuration work together.
- Reference specific models for performing web deployment tasks:
  - Creating a deployable function
  - Hosting the component delivered by the MATLAB programmer using J2EE and .NET Web technologies
  - Displaying arrays and matrices on a web page
  - Enabling scalability through stateless services
- Deploying applications through implementation of SOAP web services

Who Should Use This Guide?

This guide is organized around the different roles required in deploying MATLAB functions over the web:

- MATLAB programmer
- business services developer
- front-end web developer
- end user, who consumes the final product

For more information on these roles and skill sets and how they work together to successfully deploy a web application, see the section “Lifecycle of a Deployed MATLAB Web Application” on page 2-12.

Commonly Used Software and Preliminary Setup Information

In this section...

MATLAB Programmer

- MATLAB
- Financial Toolbox™
- MATLAB Compiler SDK

Note Many of the examples in this guide use the software listed here. It is not likely you will use all of the software listed here.

Integration Experts (Middle-Tier Developer and Front-End Web Developer)

- MATLAB Runtime
- Microsoft® IIS 5
- Microsoft .NET Framework
- Java® SDK (Software Developer Kit) 1.5 or later
- Java JRE™ (Java Runtime Environment) 1.5 or later
- Apache™ Tomcat™ 5 Web Server
- Apache Axis2™ Web Services Engine
- PHP server-side hypertext preprocessor 5.2.3 or later
- NUSOAP PHP class add-in

End User

- Microsoft Office 2003 or later
- Microsoft Office Web Services Toolkit

2

Web 101: An Introduction to Web Concepts and Terminology

Basics of Web Processing

In this section...

Integrating MATLAB Code on the Web Using Java Servlets

A Java servlet is an object that resides on the server. When a servlet is first placed on a web server, it is assigned patterns of URLs which it can process.

The servlet has a method on it, called `doGet`. `doGet` is invoked when a client issues the `get` command for any document matching the pattern assigned to a particular servlet. The `get` command is one of the most common in the HTTP protocol: it initiates the reading of a document.

The `doGet` command is passed two objects: one that represents the request and one that represents the response. `doGet` must analyze the request object for what is needed and use the response object to send back the document it creates to the browser:

Integrate your MATLAB code as follows:

1. Override the default `doGet()` method provided by the `HttpServlet` class so that it makes calls to your MATLAB code.
2. Write code to respond to the client request. At minimum, you must at least set the content type.
3. Write your data to a virtual file that is either being buffered or being sent directly to the client via socket.

   The stream output commands in your servlet include the output from calling the MATLAB function on your component. For example, you might use the `MWArray toString` method to convert your `MWArray` objects to strings or another type appropriate to your application.

Using JavaServer Pages (JSP) to Integrate MATLAB Code On the Web

Under some circumstances, you can achieve desired end result faster and more seamlessly by using JavaServer Pages (JSP). JSP is a server side Java technology that

allows software developers to create dynamically generated web pages, with HTML, XML, or other document types, in response to a web client request to a Java Web Application container.

MyServlet.java

```
import com.mathworks.toolbox.javabuilder.MWException;
import com.mathworks.toolbox.javabuilder.MWArray;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
import java.io.IOException;
import java.io.PrintStream;

import magicsquare.magic;

/**
 * A simple servlet to call MATLAB code.
 */
public class MyServlet extends HttpServlet
{
    @Override
    protected void doGet(final HttpServletRequest request,
            final HttpServletResponse response)
            throws ServletException, IOException
    {
        try {
            final magic m = new magic();
            try {
                final Object[] results = m.makesqr(1, (int)(Math.random()
                                                            * 7 + 3));
                try {
                    final MWArray firstResult = (MWArray)results[0];
                    final double[][] square = (double[][])firstResult.toArray();
                    response.setContentType("text/html");
                    final PrintStream page =
                         new PrintStream(response.getOutputStream());
                    page.println("<html>");
                    page.println("<head>");
                    page.println(" <title>Magic Square Web App</title>");
                    page.println("</head>");
                    page.println("<body>");
                    page.println("A magic square of size " + square.length);
                    page.println("<table border='1' cellpadding='4'>");
                    for (double[] row : square) {
                        page.println("<tr>");
                        for (double column : row) {
                            page.println("<td>" + column + "</td>");
                        }
                        page.println("</tr>");
                    }
                    page.println("</table>");
```

```
                    page.println("</body>");
                    page.println("</html>");
                } finally {
                    MWArray.disposeArray(results);
                }
            } finally {
                m.dispose();
            }
        } catch (MWException e) {
            throw new ServletException(e);
        }
    }
}
```

Optimizing Performance of Servlets That Call Your MATLAB Web Code

`doGet` is invoked every time a `get` command is received on the server. Thus, every time a `get` command is received,`MagicWeb` is instantiated along with its `dispose` method.

This can quickly impact performance, as it takes time to create a MATLAB Runtime instance. You can, however, take measures to improve the performance of your code.

Moving the ownership of the component from the `doGet` method into the servlet, so it doesn't have to be created and destroyed repeatedly, is a start.

Using HTTPServlet init and destroy Methods

`HTTPServlet` has a set of methods called `init` and `destroy`. The servlet automatically calls these methods when the servlet is created and destroyed.

To take advantage of these methods, you match the scope of `MagicWebJavaApp` to the scope of the servlet. This renders the lifetime of the deployed component and its MATLAB Runtime instance as the same as the lifetime of the servlet. From the first time a user requests a document that matches the pattern assigned to the servlet, until the time the servlet is terminated by an administrator, the servlet remains available.

1. Add a property on `MyServlet` of type `magic` and instantiate.

   Tip As a best practice, make the instance `private`. Doing so promotes data security and integrity.

2. Overload the `init` method (where you `create` your servlet).

3. Overload the `destroy` method (where you issue `dispose`).
4. Verify your `doGet` method in your servlet is reentrant—meaning that it can be entered simultaneously on multiple threads and processed in a thread-safe manner.

Note A Web server that uses servlets to process requests can be termed a servlet container. The servlet container manages every object scoped to its class. In this case, the container manages the lifetime of servlets by calling `init` and `destroy` for you. You specify to the container how you want `init` and `destroy` to be executed.

MyServlet.java

```
 public class MyServlet extends HttpServlet
{
    private magic m;

    @Override
    public void init(ServletConfig servletConfig) throws ServletException {
        try {
            m = new magic();
        } catch (MWException e) {
            throw new ServletException(e);
        }
    }

    @Override
    protected void doGet(final HttpServletRequest request,
            final HttpServletResponse response)
            throws ServletException, IOException
    {
        try {
            final Object[] results = m.makesqr(1, (int)(Math.random() * 7 + 3));
            try {
                final MWArray firstResult = (MWArray)results[0];
                final double[][] square = (double[][])firstResult.toArray();
                response.setContentType("text/html");
                final PrintStream page =
                      new PrintStream(response.getOutputStream());
                page.println("<html>");
                page.println("<head>");
                page.println(" <title>Magic Square Web App</title>");
                page.println("</head>");
                page.println("<body>");
                page.println("A magic square of size " + square.length);
                page.println("<table border='1' cellpadding='4'>");
                for (double[] row : square) {
                    page.println("<tr>");
                    for (double column : row) {
                        page.println("<td>" + column + "</td>");
                    }
                    page.println("</tr>");
```

```
                }
                page.println("</table>");
                page.println("</body>");
                page.println("</html>");
            } finally {
                MWArray.disposeArray(results);
            }
        } catch (MWException e) {
            throw new ServletException(e);
        }
    }

    @Override
    public void destroy() {
        m.dispose();
    }
}
```

Maintaining Data Integrity Through Proper Scoping

When working with servlets, you must be extremely aware of the role scoping plays in placement of your data.

A scope is a binding of variable to value. In the same scope, a certain name always refers to a particular binding value.

Every method call generates its own scope. Local variables belong to the scope of the method invocation.

There are several degrees of scope provided by the Java language:

- **Static scope** — Bound to the class itself
- **Instance scope** — Bound to an instance of the class
- **Method scope** — Bound to local variables tied to a specific method invocation

When you refer to a variable, the default behavior of the Java compiler is to first interpret that variable in terms of its most local scope. For example, in `MyOtherServlet.java` on page 2-7, note how variables are determined within the scopes designated by each set of curly braces.

In this example, notice that state which is particular to an individual request is scoped to the `doGet` method itself, rather than the servlet.

To ensure processing is thread-safe, avoid storing data related to state variables as instance variables on the class itself. Defining too many instance fields in the class allows

state variables to be shared by all threads that call `doGet` on the same servlet instance. Instead, scope at the method level, using method parameters, rather than at the class level. This scoping at the method level also ensures your application scales easily as more servers are added.

Context Scoping with Servlets

As discussed in “Optimizing Performance of Servlets That Call Your MATLAB Web Code” on page 2-4, `MyServlet` takes a request and generates an HTML document calling `MagicWebJavaApp`. This call generates a formatted response. However, you may want a different formatted response or you may want to call `MagicWebJavaApp` with another method to achieve a different result. To do this, you write `myOtherServlet`.

Begin by using the code from `MyServlet` as a template for `myOtherServlet`. This time, however, you install `myOtherServlet` in the servlet container by mapping it to another set of URLs.

Clients periodically do `GET`s on various URL patterns mapped to each servlet. While this works well, you can make the process more efficient—note that you are duplicating work by using two copies of `MagicWebJavaApp`.

Because each of the servlets gets their `init` method called separately, each of these methods initializes *a separate MATLAB Runtime instance*.

MyOtherServlet.java

```
public class MyOtherServlet extends HttpServlet
{
    private magic m;

    @Override
    public void init(ServletConfig servletConfig) throws ServletException {
        try {
            m = new magic();
        } catch (MWException e) {
            throw new ServletException(e);
        }
    }

    @Override
    protected void doGet(final HttpServletRequest request,
            final HttpServletResponse response)
            throws ServletException, IOException
    {
        try {
            final Object[] results = m.makesqr(1, (int)(Math.random()
                                                    * 10 + 11));
            try {
```

```
                final MWArray firstResult = (MWArray)results[0];
                response.setContentType("text/html");
                {
                    final double[][] square = (double[][])firstResult.toArray();
                    {
                        final PrintStream page =
                                 new PrintStream(response.getOutputStream());
                        page.println("<html>");
                        page.println("<head>");
                        page.println("
                         <title>Magic Square Web App (Other)</title>");
                        page.println("</head>");
                        page.println("<body>");
                        page.println("A magic square of size " + square.length);
                        page.println("<table border='1' cellpadding='4'>");
                        {
                            for (double[] row : square) {
                                page.println("<tr>");
                                for (double column : row) {
                                    page.println("<td>" + column + "</td>");
                                }
                                page.println("</tr>");
                            }
                        }
                        page.println("</table>");
                        page.println("</body>");
                        page.println("</html>");
                    }
                }
            } finally {
                MWArray.disposeArray(results);
            }
        } catch (MWException e) {
            throw new ServletException(e);
        }
    }

    @Override
    public void destroy() {
        m.dispose();
    }
}
```

Improving Performance with Servlet Contexts

Context scopes provided by the servlet container can supply convenient solutions to performance problems such as those described in “Context Scoping with Servlets” on page 2-7.

Scopes including multiple servlets are called servlet contexts. Using a servlet context, you make the component a shared resource and improve performance substantially.

1. Obtain the servlet context by calling `getServletContext` .
2. After you obtain the servlet context, get an attribute (in this case, variables bound to the servlet context) on the context.

The following logic (shown in **bold** type) added to each servlet (`MyServlet.java` and `MyServletContextListener.java`), creates a shared `MagicWebInstance`:

MyServletContextListener.java

```
import magicsquare.magic;

import javax.servlet.ServletContextListener;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContext;

import com.mathworks.toolbox.javabuilder.MWException;

public class MyServletContextListener implements ServletContextListener {
    static final String SHARED_MAGIC_INSTANCE = "SharedMagicInstance";

    public void contextInitialized(final ServletContextEvent event) {
        final ServletContext context = event.getServletContext();
        try {
            final magic m = new magic();
            context.setAttribute(SHARED_MAGIC_INSTANCE, m);
        } catch (MWException e) {
            context.log("Error creating magicsquare.magic", e);
        }
    }

    public void contextDestroyed(final ServletContextEvent event) {
        final ServletContext context = event.getServletContext();
        final magic m = (magic)context.getAttribute(SHARED_MAGIC_INSTANCE);
        m.dispose();
    }
}
```

Code Excerpt from MyServlet.java

```
  @Override
    protected void doGet(final HttpServletRequest request,
            final HttpServletResponse response)
            throws ServletException, IOException
    {
        final HttpSession session = request.getSession();
        final ServletContext context = session.getServletContext();
        final magic m =
          (magic)context.getAttribute
               (MyServletContextListener.SHARED_MAGIC_INSTANCE);
```

Session Scoping with Servlets

A Java session is defined as a lasting connection between a user (or user agent) and a peer, typically a server, usually involving the exchange of many packets between the user's computer and the server.

The servlet container provides a scope that corresponds to a session between a particular client and a server. Since HTTP is a very lightweight protocol and carries no notion of persistence, it follows that session scoping is particularly useful in retaining data or state particular to a client. If you have state that needs to persist across `request`s, the proper context is the `Session`.

Single instances of a servlet may be used for many clients throughout its lifetime. Thus, `init` and `destroy` have no knowledge of the session context. However, HTTP `request` and `response` objects do, since `request` and `response` happen within the context of a session between a client and server.

You usually get the `HTTPSession` object from the request.

`Session` has an API with similar methods to servlet context—`setAttribute`, `getAttribute`, `removeAtrribute`, and so on. They bind variables that last for the relationship between a specific client with browser and server.

Sessions usually have time-outs, so they automatically clean up in case the client doesn't explicitly log off. This is configurable on the `HTTPSession` object itself or somewhere in the servlet API.

The following logic (shown in **`bold`** type) added to both `MyHttpSessionListener.java` and `MyServer.java` creates your MATLAB component at the `session` scope.

MyHttpSessionListener.java

```
import magicsquare.magic;

import javax.servlet.http.HttpSessionListener;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpSessionEvent;

import com.mathworks.toolbox.javabuilder.MWException;

public class MyHttpSessionListener implements HttpSessionListener {
    static final String SHARED_MAGIC_INSTANCE = "SharedMagicInstance";

    public void sessionCreated(final HttpSessionEvent event) {
        final HttpSession session = event.getSession();
        try {
```

```
            final magic m = new magic();
            session.setAttribute(SHARED_MAGIC_INSTANCE, m);
        } catch (MWException e) {
            session.getServletContext().log
                   ("Error creating magicsquare.magic", e);
        }
    }

    public void sessionDestroyed(final HttpSessionEvent event) {
        final HttpSession context = event.getSession();
        final magic m = (magic)context.getAttribute(SHARED_MAGIC_INSTANCE);
        m.dispose();
    }
}
```

MyServer.java

```
   @Override
    protected void doGet(final HttpServletRequest request,
            final HttpServletResponse response)
            throws ServletException, IOException
    {
        final HttpSession session = request.getSession();
        final magic m =
           (magic)session.getAttribute
                    (MyHttpSessionListener.SHARED_MAGIC_INSTANCE);
```

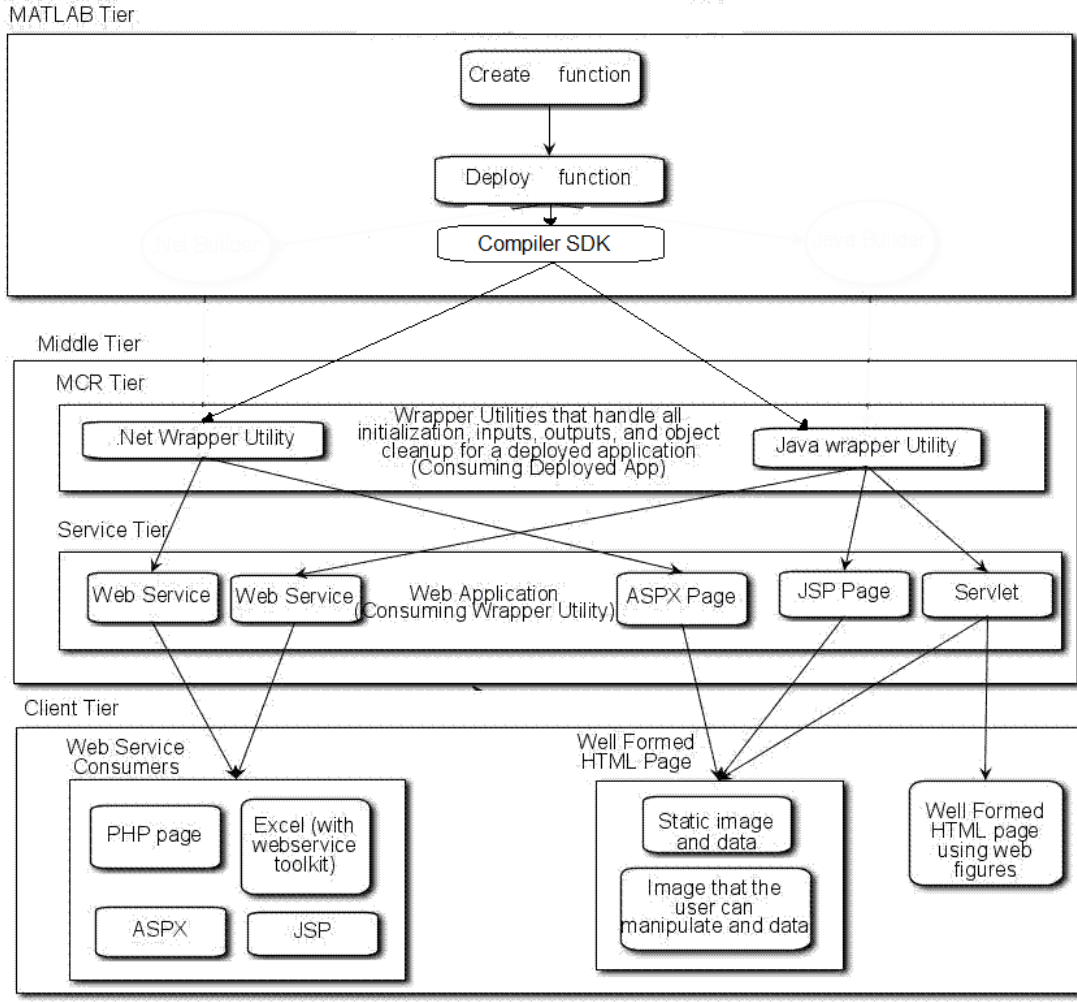
Lifecycle of a Deployed MATLAB Web Application

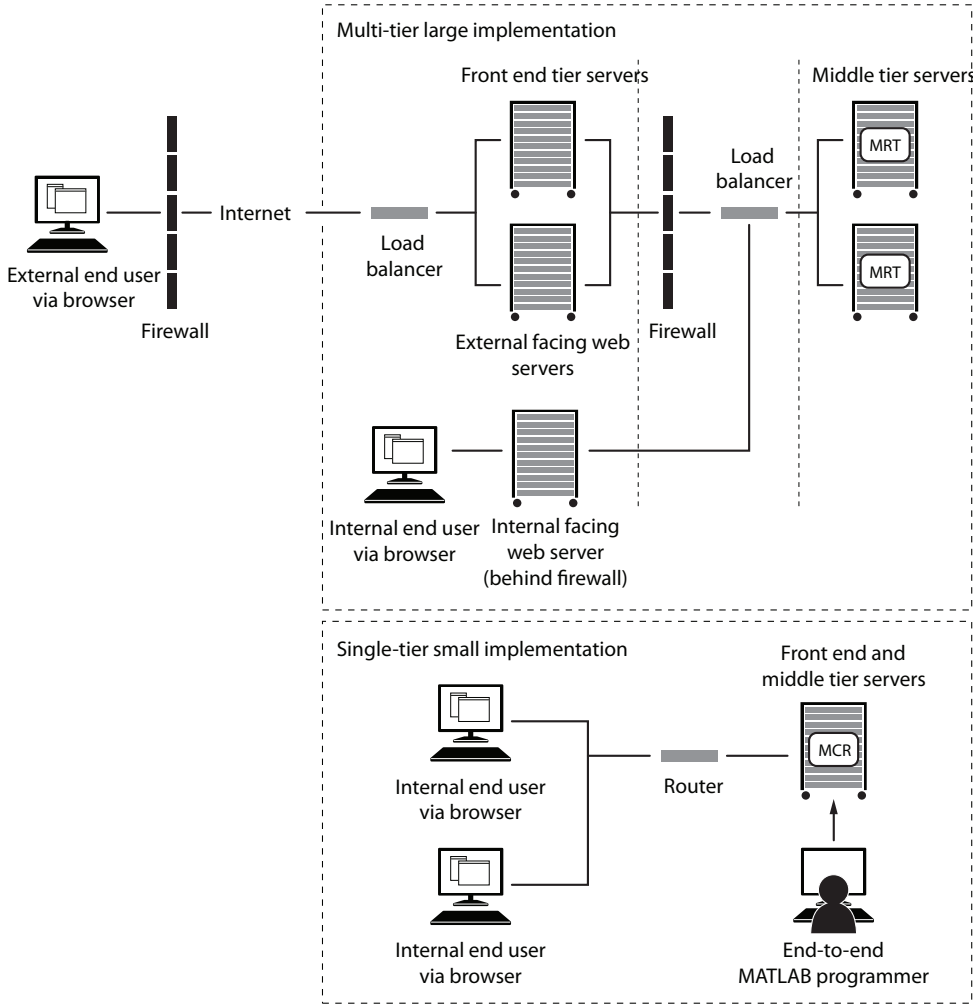
In this section...

Introduction

How does a single piece of MATLAB code become a deployable, portable, robust, scalable web application? Through skillful deployment by a number of people in an organization, each playing distinct and significant roles.

The following diagrams depict the supported implementation and architectures available when using MATLAB application deployment products.


MATLAB Tier
Create function
Deploy function
Compiler SDK
Middle Tier
MCR Tier
.Net Wrapper Utility
Wrapper Utilities that handle all initialization, inputs, outputs, and object cleanup for a deployed application (Consuming Deployed App)
Java wrapper Utility
Service Tier
Web Service
Web Service
Web Application (Consuming Wrapper Utility)
ASPX Page
JSP Page
Servlet
Client Tier
Web Service Consumers
PHP page
Excel (with webservice toolkit)
ASPX
JSP
Well Formed HTML Page
Static image and data
Image that the user can manipulate and data
Well Formed HTML page using web figures



MATLAB Web Application Environment

The fundamental goal of the MATLAB Compiler SDK product is to enable MATLAB functions to be deployed outside the MATLAB environment. This is accomplished with the MATLAB Runtime, which is a set of libraries that runs encrypted MATLAB code.

In a web application, the MATLAB Compiler SDK product allows integration of the MATLAB Runtime at the server tier level. This enables end users to execute MATLAB applications over the web without installing client software.

WebFigures is a client and server technology that further extends this capability by enabling end users to interact with a MATLAB figure in much the same way as they use an axis within MATLAB. The WebFigures functionality of MATLAB Compiler SDK allows users limited to Web access the ability to dynamically interact with MATLAB figures.

MATLAB Programmer

The first phase in a deployed application's life begins when code is written in MATLAB by a MATLAB programmer. The MATLAB programmer uses MATLAB Compiler SDK to deploy the MATLAB code for use outside of MATLAB.

MATLAB Compiler SDK encrypts MATLAB functions and wraps them in Java or .NET interfaces. The MATLAB programmer takes these deployable components and gives them to the middle-tier developer.

Middle-Tier Developer

Middle-tied developers take deployed MATLAB code and integrates it with the necessary business logic on the web server. The middle-tier developer is also responsible for configuring the web server to use the proper version of the MATLAB Runtime. When the Java or .NET component is called, it will instantiate the MATLAB Runtime to execute the underlying MATLAB code. Once these services are exposed client developers can connect to them and use them.

Client Developer

Client developers are typically responsible for user-visible functionality and know little about under-the-covers implementation. Their primary concern is the stability and security of the organization's data within the confines of a firewall. Once the client developers create some mechanism for exposing the application functionality to the end user, it is up to the end user to complete the lifecycle by interacting with the application to perform some task or solve some business problem.

Server Administrator

Server administrators are responsible for keeping the servers up and running, meeting the IT department's commitments to the rest of the organization as outlined in SLA agreements. They are not MATLAB experts and may or may not know much about integrating deployed applications in various computing environments. However, they are expert in understanding which versions of which computing environments (JREs and .NET frameworks, for example) can co-exist and run stably in order to achieve the end-user's desired results.

End User

End users may use the web site or may interact with the business tier directly. In this case, an example of a common activity would be when a financial analyst accesses a business tier web service and a complex Microsoft Excel® model. Or, they access an internal web site, performing specific tasks not available to their customers.

End-To-End Developer

End-to-end developers have the skills to implement all phases of a web deployment. They are MATLAB experts and are also skilled middle-tier developers and client developers. To this end, this guide presents examples of comprehensive deployment tasks scoped specifically to the time and resource constraints typically faced by end-to-end developers.

3

MATLAB Programmer Tasks

Programming in MATLAB

MATLAB is an interpreted programming environment. You can execute functions directly at the command prompt or through an editor in saved files. Methods may be created, having their own unique inputs and outputs. When deploying MATLAB code to other programming environments, such as .NET and Java, you must contain your MATLAB code within functions. MATLAB Compiler SDK does not allow you to use inline scripts.

For example, a simple function that adds two numbers takes two inputs and returns the result.

```
function result = addSomeNumbers(number1, number2)
    result = number1 + number2;
end
```

Deploying MATLAB Code with the MATLAB Compiler SDK Product

Writing the MATLAB code is only the first step when deploying an application. You must next determine how the application is structured. Although you might have a large amount of MATLAB code that needs to run within a component, typically only a small number of entry points need to be exposed to the calling application. It is best to determine these entry points and to make sure all inputs and outputs are necessary before deploying a Web application. The best practice is to ensure the MATLAB files that contain methods have the same name as the MATLAB file for all entry points.

4

Middle-Tier Developer Tasks

Creating a Data Access Object for Deployment

To access business objects in .NET and Java environments, you must write a data access class or classes.

The code in these examples represents what exists within the data access section of an application since it bridges across MATLAB data and data types and Java and .NET data types.

Note In these examples, a fake component generated using the MATLAB Compiler SDK product called `deploymentExamples` is used. Assume it has been imported.

Initializing a Component

Use these examples as a framework for initializing a component.

Java

```
DeploymentExamples deployment = null;
try
{
        deployment = new DeploymentExamples ();

        //**************
        //Use the deployment code here
        //  (see examples below)
        //**************
}
catch(MWException mw_ex)
{
        mw_ex.printStackTrace();
}
finally
{
        deployment.dispose();
}
```

.NET

```
DeploymentExamples.DeploymentExamples deployment = null;
try
```

```
{
        deployment = new DeploymentExamples.DeploymentExamples();

        //*****************************
        //**Use your deployment code here
        //** (See examples below)
        //*****************************
}
finally
{
        deployment.Dispose();
}
```

Interacting with a Component

You interact with a component by passing inputs to a deployed application or producing MATLAB output from a deployed application. All of these examples fit where the comment block resides in “Initializing a Component” on page 4-2and the same component class is used. The MATLAB Compiler SDK infrastructure handles data marshaling when passing parameters to a component. Data conversion rules can be found in the MATLAB Compiler SDK documentation. If a specific data type is required, you can use the `MWArray` objects and pass in the appropriate data type.

Converting an Integer to a MATLAB Data Type

Some of the ways to pass inputs to a deployed applications are demonstrated in these examples:

Java

```
int n = 3;
MWNumericArray x = new MWNumericArray(n, MWClassID.DOUBLE);
```

.NET

```
int n = 3;
MWNumericArray x = new MWNumericArray(n, true);
```

Converting Array Data to a MATLAB Data Type

Arrays can be converted to several different MATLAB data types. An example of converting a String array into a cell array follows:

Java

```
//Create the array of data…
String[] friendsArray1 =
{
        "Jordan Robert",
        "Mary Smith",
        "Stacy Flora",
        "Harry Alpert"
};

int numberOfArrayElements1 = friendsArray1.length;
int numberOfArrayColumns1 = 1;

//Create the MWCellArray to store the data
MWCellArray cellArray1 =
        new MWCellArray(
                numberOfArrayColumns1,
                numberOfArrayElements1);

//Iterate through the array and add the elements to
//    the cell array.

for(int i = 1; i<friendsArray1.length+1; i++)
{
        cellArray1.set(i, friendsArray1[i-1]);
}
```

.NET

```
String[] array =
{
        "Jordan Robert",
        "Mary Smith",
        "Stacy Flora",
        "Harry Alpert"
};

int numberOfArrayElements = array.Length;
int numberOfArrayColumns = 1;

MWCellArray cellArray =
        new MWCellArray(
                numberOfArrayColumns,
```

```
                numberOfArrayElements);
for (int i = 1; i < array.Length + 1; i++)
{
        cellArray[i] = array[i - 1];
}
```

Converting a List to a MATLAB Data Type

A list can be converted to several different MATLAB data types. An example of converting a List of Strings into a cell array follows:

Java

```
//Create a list of data…
List friendsList = new LinkedList();
friendsList.add("Jordan Robert");
friendsList.add("Mary Smith");
friendsList.add("Stacy Flora");
friendsList.add("Harry Alpert");

int numberOfListElements = friendsList.size();
int numberOfListColumns = 1;

//Create a MWCellArray to store the data
MWCellArray cellArray2 =
        new MWCellArray(
                numberOfListColumns,
                numberOfListElements);

//Iterate through the list adding the elements
//     to the cell array.

Iterator friendsListItr = friendsList.iterator();
for(int i = 1;friendsListItr.hasNext(); i++)
{
        String currentFriend = (String)friendsListItr.next();
        cellArray2.set(i, currentFriend);
}
```

.NET

```
List<String> list = new List<String>();
list.Add("Jordan Robert");
```

```
list.Add("Mary Smith");
list.Add("Stacy Flora");
list.Add("Harry Alpert");

int numberOfArrayElements = list.Count;
int numberOfArrayColumns = 1;

MWCellArray cellArray =
        new MWCellArray(
                numberOfArrayColumns,
                numberOfArrayElements);
int i = 1;
foreach (String currentElement in list)
{
        cellArray[i] = currentElement;
        i++;
}
```

Converting Name Value Pairs to a MATLAB Data Type

Java (Maps)

It is common to have maps of data (name value pairs). The corresponding .NET data type is Dictionary. The most similar data type in MATLAB is the structure. Here is an example where you convert a map of people's names into a MATLAB structure.

```
//First we create a Java HashMap (java.util.HashMap).
Map firendsMap = new HashMap();
friendsList.put("Jordan Robert", new Integer(3386));
friendsList.put("Mary Smith", new Integer(3912));
friendsList.put("Stacy Flora", new Integer(3238));
friendsList.put("Harry Alpert", new Integer(3077));

//Now we set up the MATLAB Structure that we will fill
//   with this data.

int numberOfElements = firendsMap.size();
int numberOfColumns = 1;
String[] fieldnames = {"name", "phone"};
MWStructArray friendsStruct =
        new MWStructArray(
                numberOfElements,
                numberOfColumns,
                fieldnames);
```

```
//Now we iterate through our map, filling in the structure

Iterator friendsMapItr = friendsMap.keySet().iterator();
for(int i = 1; friendsMapItr.hasNext(); i++)
{
        String key = (String)friendsMapItr.next();
        friendsStruct.set(fieldnames[0], i,
                          new MWCharArray(key));
        friendsStruct.set(fieldnames[1], i (Integer)
                          friendsMap.get(key));
}
```

.NET (Dictionaries)

```
Dictionary<String, int> dictionary =
               new Dictionary<String, int>();
dictionary.Add("Jordan Robert", 3386);
dictionary.Add("Mary Smith", 3912);
dictionary.Add("Stacy Flora", 3238);
dictionary.Add("Harry Alpert", 3077);

int numberOfElements = dictionary.Count;
int numberOfColumns = 1;
String[] fieldnames = { "name", "phone" };
MWStructArray output =
        new MWStructArray(
                numberOfElements,
                numberOfColumns,
                fieldnames);

int i = 1;
foreach (String currentKey in dictionary.Keys)
{
        output[fieldnames[0], i] = currentKey;
        output[fieldnames[1], i] = dictionary[currentKey];
        i++;
}
```

Getting MATLAB Numerics from a Deployed Application

This code resides in the `try` block for an initialized component (see “Initializing a Component” on page 4-2).

Java

```
Object[] numericOutput = null;
MWNumericArray numericArray = null;
try
{
        numericOutput = deployment.getNumeric(1);
        numericArray = (MWNumericArray)numericOutput[0];
        int i = numericArray;
}
finally
{
        MWArray.disposeArray(numericArray);
}
```

.NET

```
MWNumericArray result = (MWNumericArray)deployment.getNumeric();
int resultInt =  result.ToScalarInteger();
```

Getting MATLAB Strings from a Deployed Application

Java

```
Object[] stringOutput = null;
MWCharArray stringArray = null;
try
{
        stringOutput = deployment.getString(1);
        stringArray = (MWCharArray) stringOutput [0];
        String s = stringArray;
}
finally
{
        MWArray.disposeArray(stringArray);
}
```

.NET

```
MWCharArray result = (MWCharArray)deployment.getString();
String resultString = result.ToString();
```

Getting MATLAB Numeric Arrays from a Component

Java

```
Object[] numericArrayOutput = null;
MWNumericArray numericArray1 = null;
try
{
        numericArrayOutput = deployment.getNumericArray(1);
        numericArray1 = (MWNumericArray)numericArrayOutput[0];
        int[] array = numericArray1.getIntData();
}
finally
{
        MWArray.disposeArray(numericArray1);
}
```

.NET

```
MWNumericArray result =
         (MWNumericArray)deployment.getNumericArray();
Double[] doubleArray =
         (Double[])result.ToVector(MWArrayComponent.Real);
```

Getting Character Arrays from a Component

Java

```
Object[] stringArrayOutput = null
MWCharArray mwCharArray = null;
try
{
        stringArrayOutput = deployment.getStringArray(1);
        mwCharArray = ((MWCharArray)stringArrayOutput[0];
        char[] charArray = new char[mwCharArray.numberOfElements()];
        for(int i = 0; i < charArray.length; i++)
        {
                char currentChar =
              ((Character)mwCharArray.get(i+1)).charValue();
                charArray[i] = currentChar;
        }
}
finally
{
```

```
        MWArray.disposeArray(mwCharArray);
}
```

.NET

```
// Note that since MWCharArray doesn't have a
//    ToVector method, it is necessary
//    to iterate through and get a single
//               dimension for the output.
//   MWCharArray result =
//            (MWCharArray)deployment.getStringArray();
char[,] resultArray = (char[,])result.ToArray();
char[] outputArray = new char[resultArray.GetLength(1)];
for (int i = 0; i < resultArray.GetLength(1); i++)
{
        outputArray[i] = resultArray[0, i];
}
```

Getting Byte Arrays from a Component

Java

```
Object[] byteOutput = null;
MWNumericArray numericByteArray = null;

try
{
        byteOutput = deployment.getByteArray(1);
        numericByteArray = (MWNumericArray)byteOutput[0];
        byte[] byteArray = numericByteArray.getByteData();
}
finally
{
        MWArray.disposeArray(numericByteArray);
}
```

.NET

```
MWNumericArray result =
                (MWNumericArray)deployment.getByteArray();
byte[] outputByteArray =
            (byte[])result.ToVector(MWArrayComponent.Real);
```

Getting Cell Arrays from a Component

Java

This example shows how to iterate through a cell array and put the elements into a list or an array:

```
Object[] cellArrayOutput = null;
MWCellArray cellArray = null;
try
{
        cellArrayOutput = deployment.getCellArray();
        cellArray = (MWCellArray)cellArrayOutput[0];

        List listOfCells = new LinkedList();
        Object[] arrayOfCells =
            new Object[cellArray.numberOfElements()];

        for(int i = 0; i < cellArray.numberOfElements(); i++)
        {
                Object currentCell = cellArray.getCell(i + 1);

                listOfCells.add(currentCell);
                arrayOfCells[i] currentCell;
        }
}
finally
{
        MWArray.disposeArray(cellArray);
}
```

.NET

```
MWCellArray result = (MWCellArray)deployment.getCellArray();

List<Object> outputList = new List<Object>();
Object[] outputArray = new Object[result.NumberOfElements];
for (int i = 0; i < result.NumberOfElements; i++)
{
        outputArray[i] = result[i + 1];
        outputList.Add(result[i + 1]);
}
```

Getting Structures from a Component

Java

```
Object[] structureOutput = deployment.getStruct(1);
MWStructArray  structureArray =
               (MWStructArray)structureOutput[0];

try
{
        Object[] structureOutput = deployment.getStruct(1);
        structureArray = (MWStructArray)structureOutput[0];

        Map mapOfStruct = new HashMap();

        for(int i = 0; i < structureArray.fieldName().length(); i++)
        {
                String keyName = structureArray.fieldNames()[i];
                Object value = structureArray.getField(i + 1);

                mapOfStruct.put(keyName, value);
        }
}
finally
{
        MWArray.disposeArray(structureArray);
}
```

.NET

```
MWStructArray result = (MWStructArray)deployment.getStruct();

Dictionary<Object, Object> output =
              new Dictionary<Object, Object>();
for (int i = 0; i < result.FieldNames.Length; i++)
{
        output.Add(result.FieldNames[i],
        result.GetField(result.FieldNames[i]));
}
```

Getting a WebFigure from a Component and Attaching It to a Page

Java

```
Object[] webFigureOutput = null;
MWJavaObjectRef webFigureReference = null;
```

```
try
{
        webFigureOutput = deployment.getWebFigure(1);
        webFigureReference = (MWJavaObjectRef)webFigureOutput[0];
        WebFigure f = (WebFigure)webFigureReference.get();
}
finally
{
        MWArray.disposeArray(webFigureOutput);
        MWArray.disposeArray(webFigureReference);
}

//forward the request to the View layer (response.jsp)
RequestDispatcher dispatcher =
        request.getRequestDispatcher("/response.jsp");
dispatcher.forward(request, response);
```

Note This code will not do anything if executed directly. It needs a `response.jsp` to produce output.

.NET

The first two lines of code will not do anything if executed directly. It needs a `WebFigureControl` on a front end page that is called `WebFigureControl1`. If you are not using a local `WebFigureControl` and want to simply return the WebFigure in the current response object, use this code, for example:

```
WebFigure webFigure = new WebFigure(deployment.getWebFigure());
WebFigureControl1.WebFigure = webFigure;

//First, attach the webfigure to one of ASP.NET's caches,
// in this case the session cache
Session["SessionStateWebFigure"] = webFigure;
//Now, use a WebFigure Utility to get an HTML String that
//  will display this figure, Notice
//  how we reference the name we used when attaching it
//  to the cache and we indicate
//  that the Attach type is session.

String localEmbedString =
        WebFigureServiceUtility.GetHTMLEmbedString(
```

```
                "SessionStateWebFigure",
                WebFigureAttachType.session,
                300,
                300);
Response.Write(localEmbedString);
```

Getting Encoded Image Bytes from an Image in a Component

Java

```
public byte[] getByteArrayFromDeployedComponent()
{
    Object[] byteImageOutput = null;
    MWNumericArray numericImageByteArray = null;
    try
    {
        byteImageOutput =
            deployment.getImageDataOrientation(
                1,      //Number Of Outputs
                500,    //Height
                500,    //Width
                30,     //Elevation
                30,     //Rotation
                "png"   //Image Format
            );

        numericImageByteArray =
              (MWNumericArray)byteImageOutput[0];
        return numericImageByteArray.getByteData();
    }
    finally
    {
        MWArray.disposeArray(byteImageOutput);
    }
}
```

.NET

```
public byte[] getByteArrayFromDeployedComponent()
{
    MWArray width = 500;
    MWArray height = 500;
    MWArray rotation = 30;
    MWArray elevation = 30;
    MWArray imageFormat = "png";
```

```
    MWNumericArray result =
        (MWNumericArray)deployment.getImageDataOrientation(
            height,
            width,
            elevation,
            rotation,
            imageFormat);
    return (byte[])result.ToVector(MWArrayComponent.Real);
}
```

Getting a Buffered Image in a Component

Java

```
public byte[] getByteArrayFromDeployedComponent()
{
    Object[] byteImageOutput = null;
    MWNumericArray numericImageByteArray = null;
    try
    {
        byteImageOutput =
            deployment.getImageDataOrientation(
                1,      //Number Of Outputs
                500,    //Height
                500,    //Width
                30,     //Elevation
                30,     //Rotation
                "png"   //Image Format
            );

        numericImageByteArray =
                 (MWNumericArray)byteImageOutput[0];
        return numericImageByteArray.getByteData();
    }
    finally
    {
        MWArray.disposeArray(byteImageOutput);
    }
}

public BufferedImage getBufferedImageFromDeployedComponent()
{
    try
    {
```

```
        byte[] imageByteArray =
               getByteArrayFromDeployedComponent()
        return ImageIO.read
                (new ByteArrayInputStream(imageByteArray));
    }
    catch(IOException io_ex)
    {
        io_ex.printStackTrace();
    }
}
```

.NET

```
public byte[] getByteArrayFromDeployedComponent()
{
    MWArray width = 500;
    MWArray height = 500;
    MWArray rotation = 30;
    MWArray elevation = 30;
    MWArray imageFormat = "png";

    MWNumericArray result =
        (MWNumericArray)deployment.getImageDataOrientation(
            height,
            width,
            elevation,
            rotation,
            imageFormat);
    return (byte[])result.ToVector(MWArrayComponent.Real);
}

public Image getImageFromDeployedComponent()
{
    byte[] byteArray = getByteArrayFromDeployedComponent();
    MemoryStream ms = new MemoryStream(myByteArray, 0,
    myByteArray.Length);
    ms.Write(myByteArray, 0, myByteArray.Length);
    return Image.FromStream(ms, true);
}
```

Getting Image Data from a WebFigure

The following example shows how to get image data from a WebFigure object. It also shows how to specify the image type and the orientation of the image.

.NET

```
WebFigure figure =
    new WebFigure(deployment.getWebFigure());
WebFigureRenderer renderer =
    new WebFigureRenderer();

//Creates a parameter object that can be changed
// to represent a specific WebFigure and its orientation.
//If you dont set any values it uses the defaults for that
// figure (what they were when the figure was created in M).
WebFigureRenderParameters param =
    new WebFigureRenderParameters(figure);

param.Rotation = 30;
param.Elevation = 30;
param.Width = 500;
param.Height = 500;

//If you need a byte array that can be streamed out
// of a web page you can use this:
byte[] outputImageAsBytes =
    renderer.RenderToEncodedBytes(param);

//If you need a .NET Image (can't be used on the web)
// you can use this code:
Image outputImageAsImage =
    renderer.RenderToImage(param);
```

5

Front-End Web Developer Tasks

Working with the Front-End Layer

In this section...

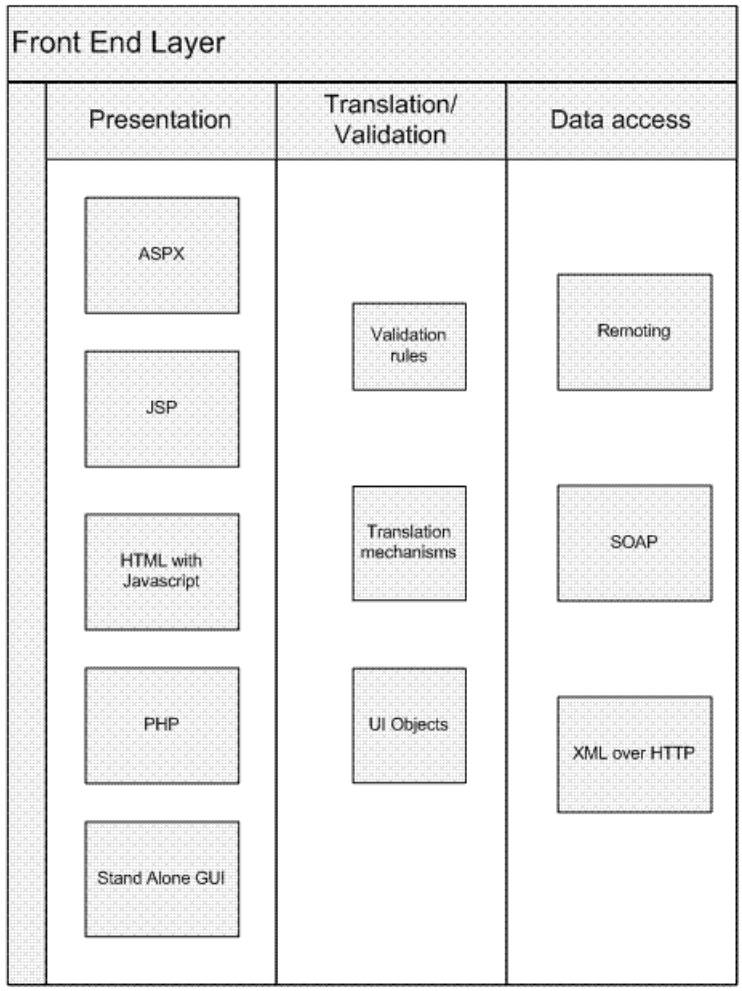
About the Front-End Layer

In well-designed multi-tier application architectures, the front-end layer presents data to the end user and validates the user's input. This is accomplished by accessing data acquired at lower-level architectural tiers to the user and taking in user inputs, validating them, and then sending them to the lower-level tiers for processing.

The data within this layer reside on servers that are almost always outside of the corporate firewall and therefore, accessible by everyone. Consequently, security and stability are integral to the front-end layer, and it is important to isolate implementation details outside of this layer so people cannot determine how your site is architected.

A well-designed front-end layer has data access, translation and validation, and presentation functions separated into individual logical code sections. This increases an application or Web site's maintainability since you can change where the data originates or the format that it arrives in without changing user-visible code.

A typical front-end layer contains the following sublayers.



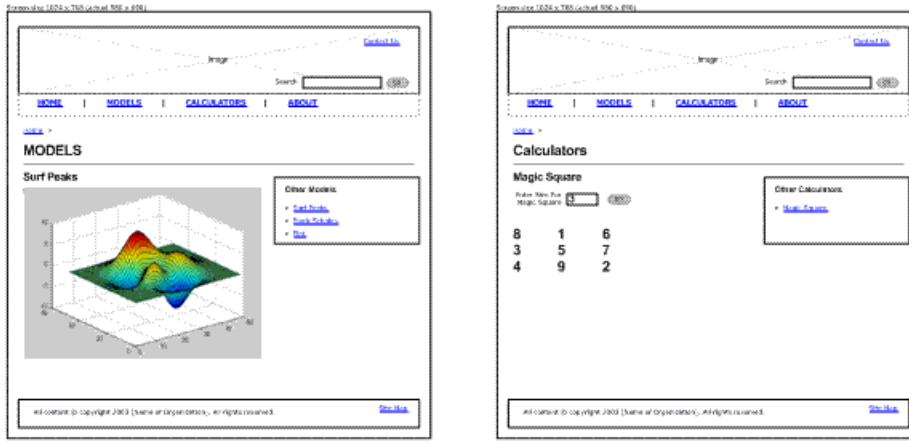
Elements of the Front End Layer

- Data Access — This sublayer pulls data in from middle-tier services like databases, where access into a deployed application would typically take place. Among the technologies used to transmit data at this sublayer are:
  - Remoting interfaces
  - SOAP services

  - XML over HTTP protocol
- Translation/Validation — Data is passed from the data access sublayer to the translation sublayer and translated into objects used for data presentation. Since these objects represent what the user sees (rather than the underlying business logic) they are very lightweight and easy to maintain. This is also where any validation would occur to ensure that values are in a proper state for processing.
- Presentation — This layer uses the data in the business objects to display information on a web site. Any user input actions are validated in the objects and, if needed, callbacks to the middle layer occur to retrieve updates based on the user input.

About the Examples

Dealing with MATLAB data is, for the most part, no different then dealing with other web data except for the fact that dynamically generated images may be involved. The examples in this document are not meant to show how to build a web site, but rather to demonstrate what types of building blocks are needed to work with MATLAB data in an existing web site. Most of these examples can be integrated directly into larger applications containing JSP, HTML, or ASPX code.



Surf Peaks and Magic Square Application Integration

The following two templates show how you can integrate applications built with MATLAB products into a larger application. In each case, there is a small area in the interface where your applications exist after the user enters input (if only a mouse click).

In the left template, it is possible to have a simple `IMG` tag, where the `src=` is a servlet from the middle tier that dumps out the image data. It is also possible to use an interactive AJAX component embedded in a subframe, or to use embedded WebFigures.

In the right template, clicking the **Go** button triggers the page to validate that the value in the input box is valid, and then sends that data to the middle tier service which returns a two-dimensional array. It is the front-end layer's job to format this data and present it properly.

In the examples that follow, these concepts are simplified and focus on how the communication occurs within the middle layer, and how some typical data translations are performed.

Displaying Complex Data Types Including Arrays and Matrices

You typically translate raw matrix array data to a form of displayable output. This section provides examples using Java and .NET.

Working with JSP Page Data

In this example, a two-dimensional array (the output of a magic square, for example) is converted to an HTML table from a JSP page. This example assumes you have gotten the data from the middle tier and have converted it back to an array.

```
<table border=0 cellpadding=4
       cellspacing=4 style='margin: 16px;'>
<%
    double[][] square = getMatrix();
 for (double[] row : square)
 {
      pageContext.getOut().print("<tr>");

    for(double value : row)
    {
            pageContext.getOut().print("<td>" + (int)value + "</td>");
    }
    pageContext.getOut().print("</tr>");
    }
%>
```

Working with ASPX Page Data

The following examples use basic ASPX pages and can be incorporated into a large enterprise site.

The easiest way to output a matrix is to iterate the array and then convert it into an HTML table. Unfortunately, this approach is not maintainable for large volumes of data, but is worth exploring in this example, assuming you have communicated with the middle tier and received a two-dimensional array of data. Assuming you have a label on the ASPX page called `MatrixLbl`, here is the code to output the matrix:

```
int size = 5;
double[][] magicSquare = getMagicSquare(size);
```

```
String temp = "";
temp +="<table border=0 cellpadding=4
          cellspacing=4 style='margin: 16px;'>";

for (int i = 0; i < size; i++)
{
        temp += "<tr>";
        for (int j = 0; j < size; j++)
        {
                temp += "<td>" + magicSquare[i][j] + "</td>";
        }
        temp += "</tr>";
}
MatrixLbl.Text = temp;
```

Using ASP.NET to Integrate with WYSIWYG Controls

ASP.NET provides a number of streamlined methods to place a grid of data on a Web page, such as mapping the data into a DataTable and referencing the DataTable from an ObjectDataSource's `Select` method. By choosing this option, you promote reuse and also maintain separation between the application's visualization and logic.

You must first place a GridView onto a page, and then "bind" it to a data source. By using an Object data source, you allow an object to dynamically get the data from some location (like a middle tier), and put it into a DataTable. Once this is done, the GridView will automatically display it.

Here is an example of using the `Select` method in your business object:

```
public DataTable getMagicSquare(int size)
{
        //Gets the matrix from the web service.
        double[][] magicSquare = getMatrix(size);

        //Create an empty data table to put the matrix data in.
        DataTable table = new DataTable();

        //Since we know its a square add as many
        //  columns as there will be rows.
        for(int i = 0; i<size; i++)
        {
                table.Columns.Add();
        }
```

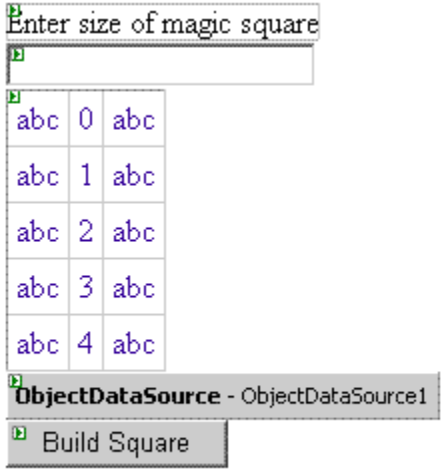
```
        DataRow row;
        //Iterate each element in the array creating a row out of each
        for(int i = 0; i < size;i++)
        {
                //create a row from the table to put the data in
                row = table.NewRow();
                //Iterate each element in the inner array and put
                // them into the row
                for (int j = 0; j < size; j++)
                {
                        row[j] = magicSquare[i][j];
                }

                //Add the row to the table
                table.Rows.Add(row);
        }
        return table;
}
```

Note If you are using graphics functions with ASP.NET, you may need to enable interaction between IIS and the desktop. For information about how to accomplish this, see the related workaround in the “Diagnostic Messages”.

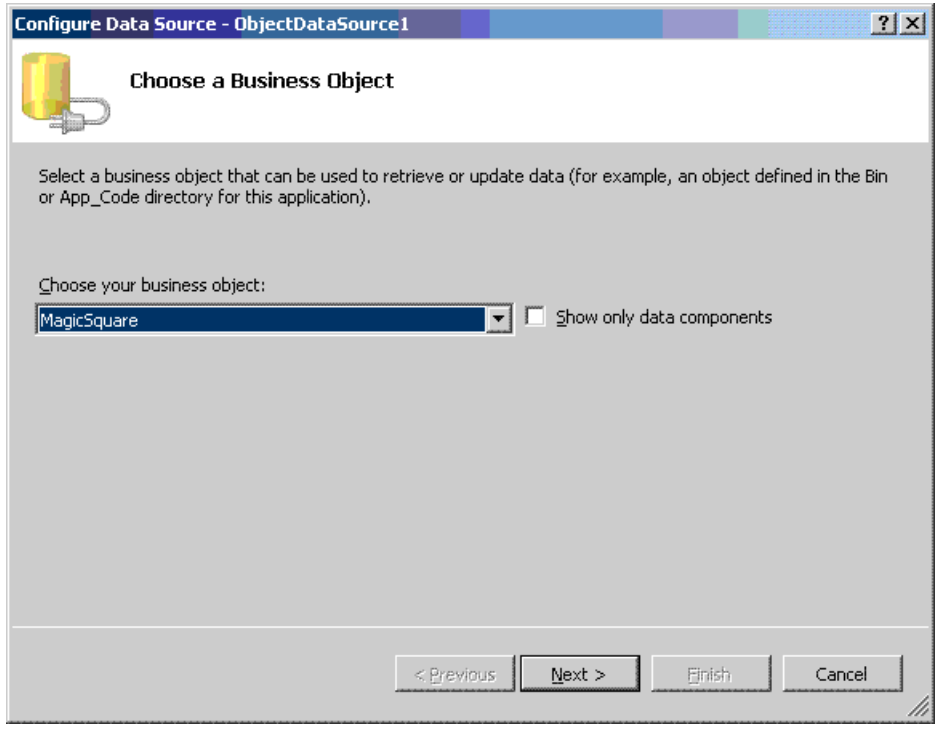
Working with ASP.NET Using the Visual Studio Wizard

This section demonstrates how to perform the implementation described in “Working With ASP.NET” when using the Visual Studio wizards and a typical Web page application. Perform the following steps to set up the data source, bind it to a load method, and bind the method's input parameter to the text box.

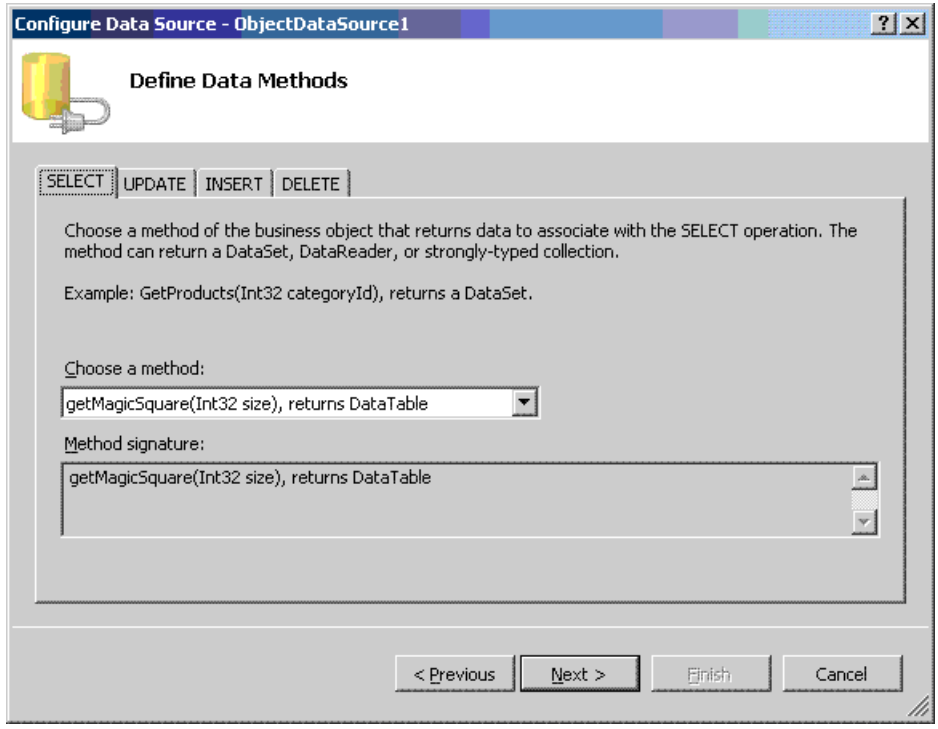
Below is an input text box and a generic grid component from a Web page. The grid component is connected to the ObjectDataSource.



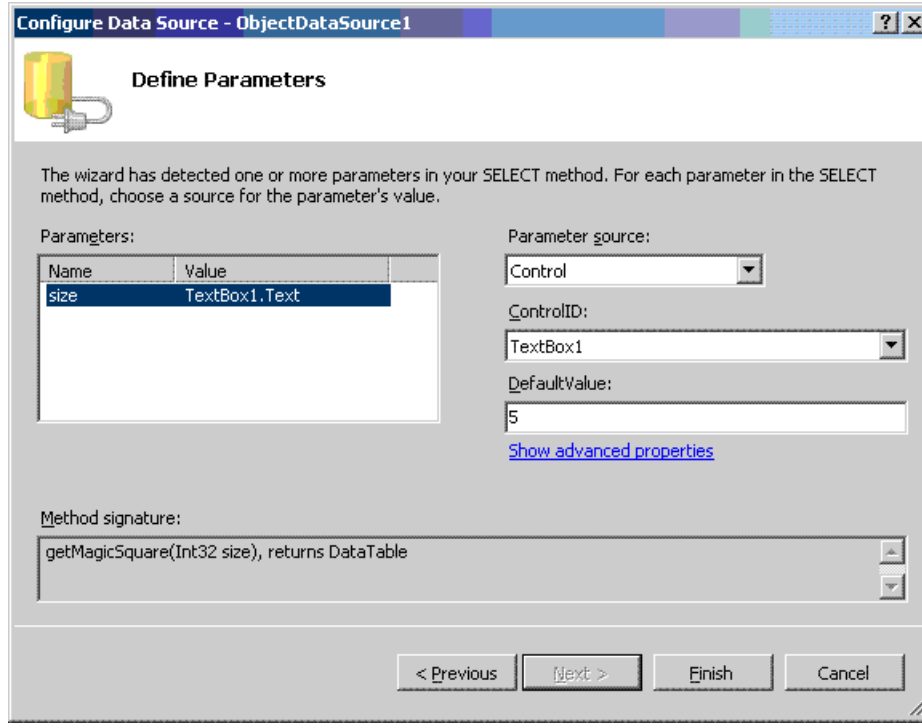
1. Start Visual Studio and configure the data source with the wizard.
2. Choose the business class that contains your methods for the object.



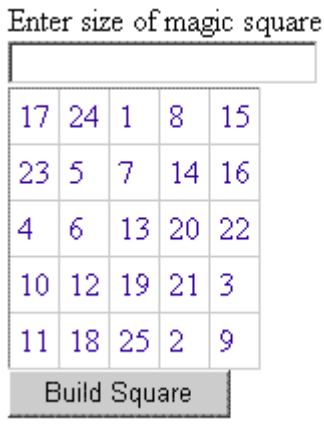
3 Select the method that returns a data table containing the data to display.



4 Since the method requires an input, bind it to the control that contains the value and set the default.



The finished application looks like this.



5 Click the **Build Square** button to reselect the grid data.

Note If you are using graphics functions with ASP.NET, you may need to enable interaction between IIS and the desktop. For information about how to accomplish this, see the related workaround in the “Diagnostic Messages”.

Using Web Services

Displaying Web Services Images and Data in PHP

If your installation has a strong investment in PHP front ends, consider using them to display Web Services running MATLAB applications.

As long as your business tier services output data in a generic nonlanguage-specific manner (as most of the examples in this document support), you can embed that output within any Web front end. This example demonstrates how to use SOAP Web services to embed an image onto a PHP page:

```
//References a soap library and loads the WSDL.
include("lib/nusoap.php");
$soapclient = new soapclient
 ('http://localhost:3465/
          SurfPeaksWebServiceServer/Service.asmx?WSDL',
                                                 true);

//If we had any parameters to pass
//  we would add them to this array.
$params = array();

//Calls the service with the parameters.
$result = $soapclient -> call("SurfPeaksWebService", $params);

//Gets the encoded responce out of the result object.
$base64EncodedResult = $result["SurfPeaksWebServiceResult"];
//Decodes and displays the result.
echo base64_decode($base64EncodedResult);

//Unloads the soap client.
unset($soapclient);
```

You can use this technique to access data services, as well:

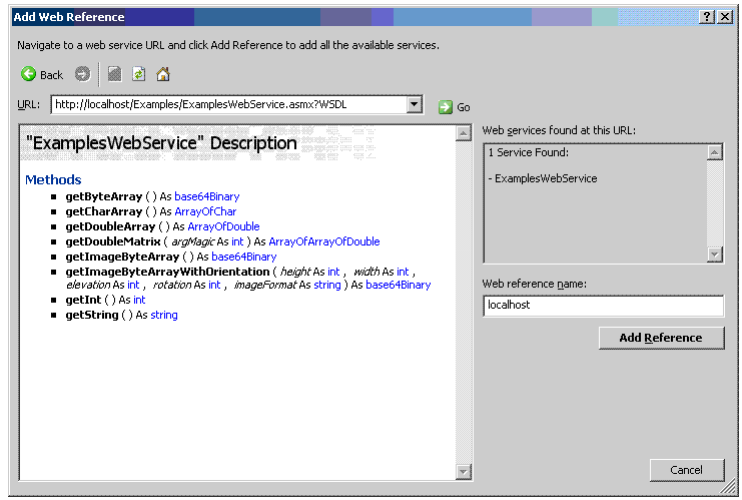
1. Install PHP 5.2.3 into IIS 5, if needed (the installer lets you specify the server type).
2. Download NUSOAP and place it on the instance path.

You should be able to use any SOAP add-in. However, note that the call syntax may change slightly. Consult the add-in documentation for further information.

Displaying Web Services Images and Data in .NET

In “Creating a Data Access Object for Deployment” on page 4-2, you exposed several methods as SOAP Web services. To use these methods from the front end, add a reference to them in your Visual Studio project by adding a **Web Reference**:

1. In Visual Studio, right-click the project name and select **Add Web Reference**.
2. In the **URL** field, enter the path to your WSDL.
3. Click **Go**. The resulting dialog resembles this.



   Notice how all of the methods exposed earlier are displayed.
4. To add these methods to your project, click **Add Reference**.
5. To use any of the methods, instantiate the Web service by executing code similar to:

   ```
   localhost.ExamplesWebService webService =
   new localhost.ExamplesWebService();
   ```
6. Access any of the methods on the Web service as you would any other .NET object. Here are some examples:

   ```
   int intValue = webService.getInt();
   string stringValue = webService.getString();
   double[] doubleArray = webService.getDoubleArray();
   double[][] magicSquare = webService.getDoubleMatrix(size);
   char[] charArray = webService.getCharArray();
   ```

```
byte[] bytesArray = webService.getByteArray();
byte[] imageByteArray = webService.getImageBytesArray();
byte[] imageByteArrayWithOrientation =
webService.getImageByteArrayWithOrientation(    500,
                                    500, 20, 30, "png");
```

6

Server Administrator Tasks

Managing a Deployment Server Environment

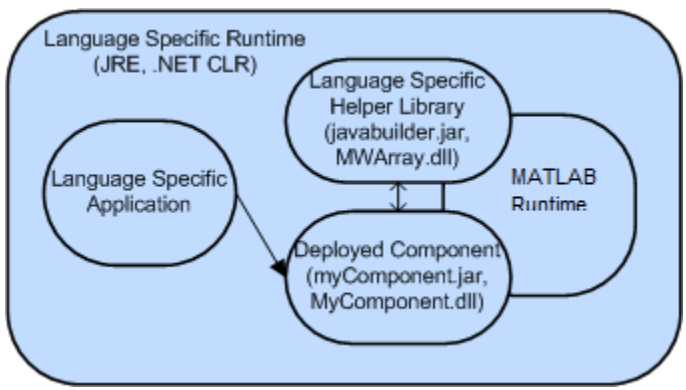
In this section...

An Overview of Deployed Applications

You can compare the relationship between compiled Java code and the JRE to the relationship between a compiled MATLAB program and the MATLAB Runtime. To execute Java code, you need a compatible version of the JRE on the system to execute it.

The deployment products, similarly, provide the compiled MATLAB application with a language-specific wrapper. The MATLAB Runtime provides the run-time infrastructure that runs deployed applications. It is made up of mostly native code.

When a deployed application integrates with a .NET or Java application, it resembles the figure “An Integrated Deployed Application.”



An Integrated Deployed Application

In this figure, you can see how the MATLAB Runtime (a large native code base) exists within the outer process of the frameworks. This coexistence can cause issues, as illustrated later in this chapter.

Installing the MATLAB Runtime

The MATLAB Runtime is a large application made up mostly of unmanaged code. It comprises an almost full copy of MATLAB, only distributable. It has no desktop and only executes encrypted code generated by the MATLAB Compiler SDK. For a deployed application to function against the MATLAB Runtime, you include references to specific directories on your Windows®, Linux®, or Mac OS X path. For more information, see End User Installation with the MATLAB Compiler Runtime (MCR) (MATLAB Compiler).

The MATLAB Runtime Installer

The MATLAB Runtime installer ships with MATLAB. You obtain it from the MATLAB programmer who originally compiled the application. The version number of the MATLAB Runtime installer is the same as the version number of MATLAB. The platform you install it on is the same as the platform on which MATLAB runs. The installer places the MATLAB libraries in a configurable location and, on Windows, automatically updates the system path.

You sometimes run a deployed component on a different platform than where it originated (especially for the Java target). To port across platforms, you need a version of the MATLAB installer for your target platform.

Helper Library Locations

For both MATLAB Compiler SDK, a Helper Library ships with the MATLAB Runtime. This library contains information for communicating with the MATLAB Runtime. It also communicates with helper utilities and data types implemented by the wrapper code. Using these wrappers, you can convert your data to and from MATLAB data types.

.NET

For .NET, the Helper Library resides at the following location:

- *mcrroot*`\toolbox\dotnetbuilder\bin\win64\v`*major*`.`*minor*`\MWArray.dll` — Contains MATLAB data type wrappers, utilities, and MATLAB Runtime communication utilities

Note v*major.minor* denotes the version of the .NET framework.

Note *mcrroot* is the MATLAB Runtime installation directory.

Both of these libraries install into the Global Assembly Cache automatically when you install the MATLAB Runtime. Although they reside in the GAC, you reference them from Microsoft Visual Studio® during development.

You can have multiple versions of this library in the GAC.

Java

For Java, the Helper Library resides at the following location:

- *mcrroot*`\toolbox\javabuilder\jar\javabuilder.jar`

Place this JAR-file on the `classpath` exactly one time for any Java application that uses a deployed application.

For a Web server, place this component in the shared library location for your server to allow all Web applications to inherit these libraries. In older versions of Tomcat, this directory is *tomcat_root*`/common/lib/`.

Caution Placing `javabuilder.jar` in the `WEB-INF/Lib` folder for a single Web application generally works. However, if another application also places `javabuilder.jar` in its `WEB-INF/Lib` locations, problems may occur. The native resources associated with `javabuilder.jar` can be loaded only once in an application. Therefore, `javabuilder.jar` must only be visible to a single class loader.

Note *mcrroot* is the MATLAB Runtime installation directory.

Loading the MATLAB Runtime

The MATLAB Runtime loads when a class in a deployed component is instantiated for the first time.

MATLAB Runtime loading can take anywhere from 10 seconds to over a minute. To ensure a consistent user experience for web applications, load the MATLAB Runtime at

server startup time, rather than upon first use, by instantiating a class as part of the server initialization.

.NET

In ASP.NET Web applications, create a Global Assembly Cache (`Global.asax`) using this code:

```
<%@ Application Language="C#" %>
<script runat="server">
       void Application_Start(object sender, EventArgs e)
       {
           // Code that runs on application startup

           myComponent.MyComponent comp =
                new myComponent.MyComponent();
       }
   </script>
```

Note You do not need to dispose of the component. You utilize the garbage collector to clean up these objects, even though they reference native components.

Java

In a J2EE server, you write a `ContextListener` class that contains code the server automatically runs when you install or remove the application.

1. Utilize the class by placing the following code in your `web.xml` file:

   ```
   <listener>
     <listener-class>myContextListenerClass</listener-class>
   </listener>
   ```

2. Place this code in your class:

   ```
   import javax.servlet.*;

   public final class myContextListener
              implements ServletContextListener
   {
       public void contextInitialized(ServletContextEvent event)
       {
          // This method is called when the application
   ```

```
            //   is first deployed in the server

            //Instantiate the deployed component,
            //  this will trigger the MATLAB Runtime to be started
            event.getServletContext().setAttribute
                    (       "myComponent",
                     new myComponent.MyComponent());
        }

        public void contextDestroyed(ServletContextEvent event)
        {
            // This method is called when the application
            //   is shut down on the server

            myComponent.MyComponent myComp =
                    event.getServletContext().
                      getAttribute("myComponent");
            if(myComp != null)
            {
                //   Dispose of the object when the
                //   server is shut down
                //   since it utilizes native resources
                //   that the garbage
                //   collector won't clean up.
                myComp.dispose();
            }
        }
    }
```

Scaling Your Server Environment

There are two methods to achieving scalability:

- “Calculation Scaling” on page 6-6
- “Session Scaling” on page 6-7

Calculation Scaling

Calculation scaling involves increasing computer resources to scale and improve performance for a specific calculation. In MATLAB, the Parallel Computing Toolbox™ enables your MATLAB code to use features built into the MATLAB language. These features tie into a profile and enable your functions to run in parallel. Parallel processing can drastically speed up the execution of a function.

There are multiple strategies towards make your applications scalable—one is done by writing your MATLAB code to scale to a parallel computing algorithm. You may ultimately have calculation scaling as well as “Session Scaling” on page 6-7 to optimize performance.

Session Scaling

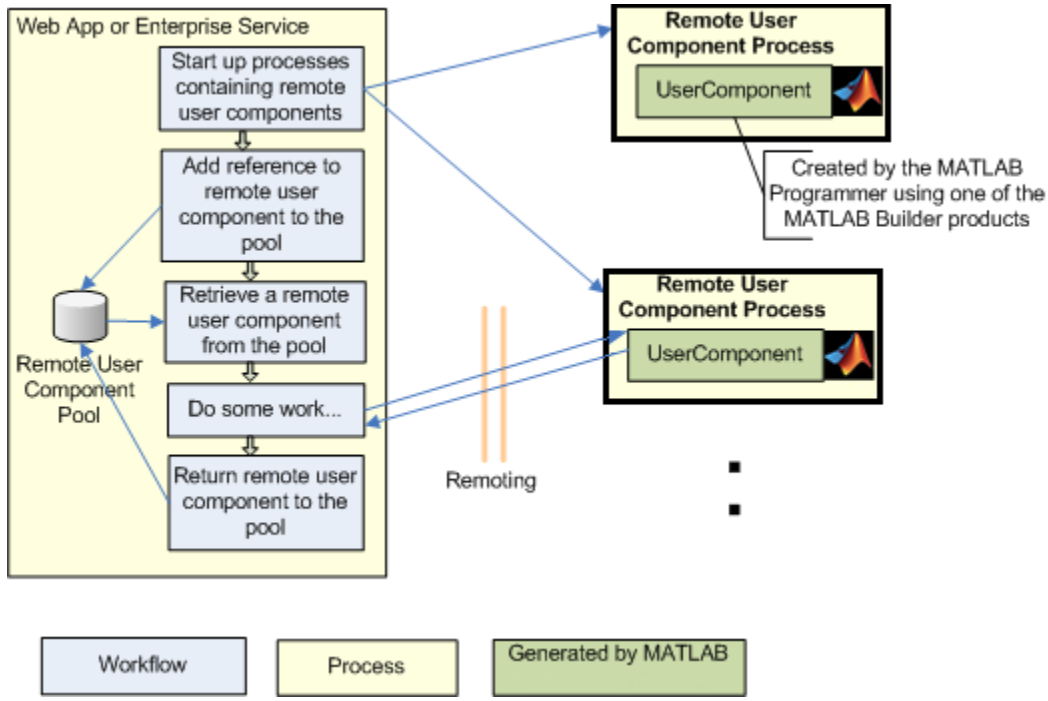
Session scaling involves enabling the maximum number of users while minimizing performance degradation.

Many session scaling issues arise because the MATLAB Runtime is single threaded. A single-threaded application prevents two users from doing work that involves the MATLAB Runtime at the same time. One user must wait for the other to finish before continuing. This wait can prove to be substantial if one user is performing a resource-intensive task while the other is attempting a quick calculation.

To work around the situation, enable multiple MATLAB Runtimes to service requests as they arrive. Run several MATLAB Runtimes in separate processes; one process per MATLAB Runtime. Using this technique, you can create a separate server process that receives requests, runs the requests against one of the processes, and returns the result.

In one solution, servers reside in a third-party grid managed by a tool that spawns processes for each instance. Alternatively, you create your own pooling solution and manage these processes manually. For either approach, you accomplish the communication using either Java RMI or “Create a Remotable .NET Assembly” because deployed components and data types can be serialized.

In most cases, MATLAB code executes quickly, and you do not need to do anything to get your desired level of performance. As a best practice, start with a single MATLAB Runtime. As your usage grows, add in a scaling layer as needed. Adding another layer involves minimal client changes.



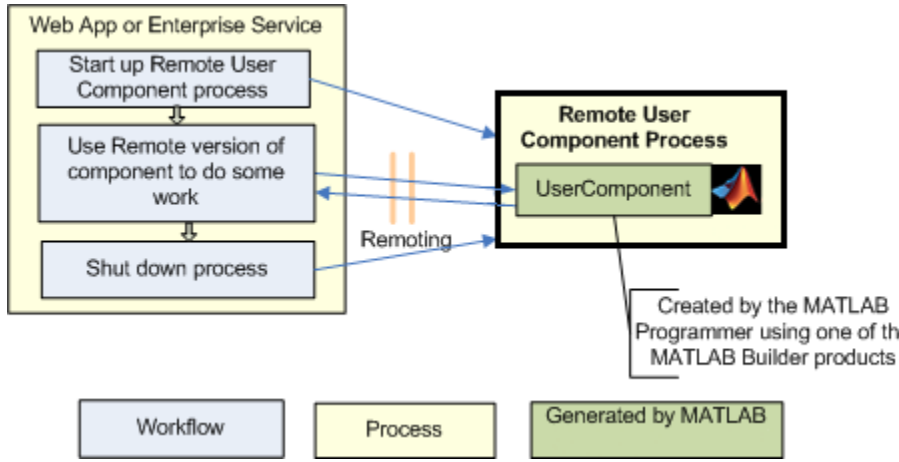
Remote Scaling

Ensuring Fault Tolerance

In your default scenario, avoid using the MATLAB Runtime in the same process as the rest of your application. The MATLAB Runtime is a large complex native application running inside your applications process. If the MATLAB Runtime executes in the same process, you cannot ensure fault tolerance in the compiled application. To ensure that it does not affect the outer process, move it to its own process and pass the data back and forth.

Remoting provides the ideal solution, as discussed in “Scaling Your Server Environment” on page 6-6. Remoting allows you to start up a process whose only job is to start the MATLAB Runtime and run requests against it. Starting this process enables lightweight access from the client.

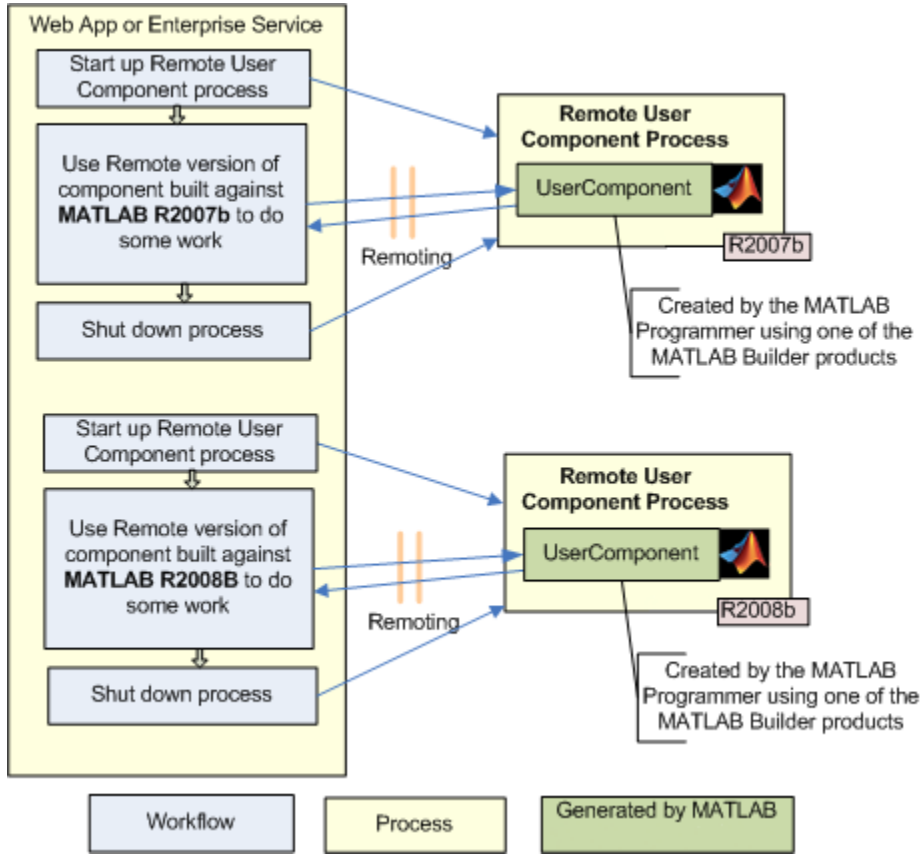
MATLAB Compiler SDK has features that allow you to auto-convert MATLAB data types into Java or .NET data types. This auto-conversion frees you from running a MATLAB Runtime where the client process executes, yielding a more robust application.



Using Remoting to Achieve Fault Tolerance

Working with Multiple Versions of the MATLAB Runtime

You can run two applications from the same Web server that link against deployed components built in different versions of MATLAB. To do so, create server processes for each application. You reference the applications, using remoting, back to the server. By using remoting, you ensure that one version of the MATLAB Runtime libraries loads into any given process.

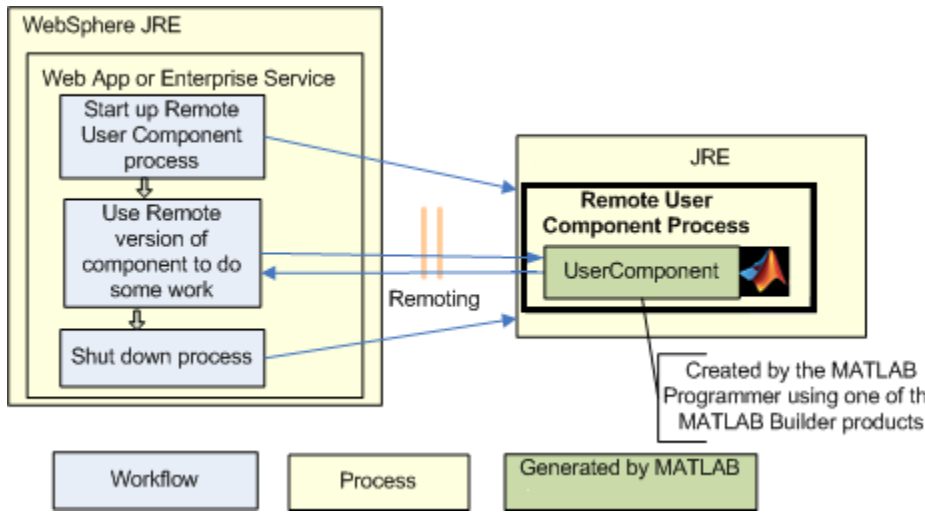


Using Remoting to Workaround Multiple Versions of the MATLAB Runtime

Unsupported Versions of the JVM

The MATLAB Runtime internally utilizes a JVM unless you disable it manually. If you are using a deployed application from within a Java application (with its own JVM), the MATLAB Runtime will attach to and use that application's JVM.

To do this successfully, you must ensure a supported MATLAB JVM version is available to your application. For example, servers such as IBM® WebSphere™ are incompatible for compiled applications because they use an IBM JVM, rather than an Oracle® JVM™, for example. You can workaround this issue by using remoting to pull the MATLAB Runtime into its own process, where it uses the proper JVM.



Using Remoting to Workaround an Unsupported JVM

End User Tasks

Working with Content

End users access the business logic through tools such as Microsoft Excel or a Web page on the front-end tier. The end user sees only the resulting data and has no need (or need to know) the implementation used to create it.

End-to-End Developer Tasks

Magic Square Calculator On the Web

The examples in this chapter demonstrate a Magic Square Calculator application that allows users to input a size for a magic square. It shows the matrix, as well as a surface plot of the matrix. This surface plot doesn't represent anything, but it demonstrates how to handle numerical data as well as visualization data.

The applications built in this chapter are not complex multi-tiered applications. Rather, these applications represent the product of the least number of steps required to build a working Web application quickly. The concepts demonstrated in this chapter can be extended into a robust, scalable Web application using techniques from other chapters in this guide.

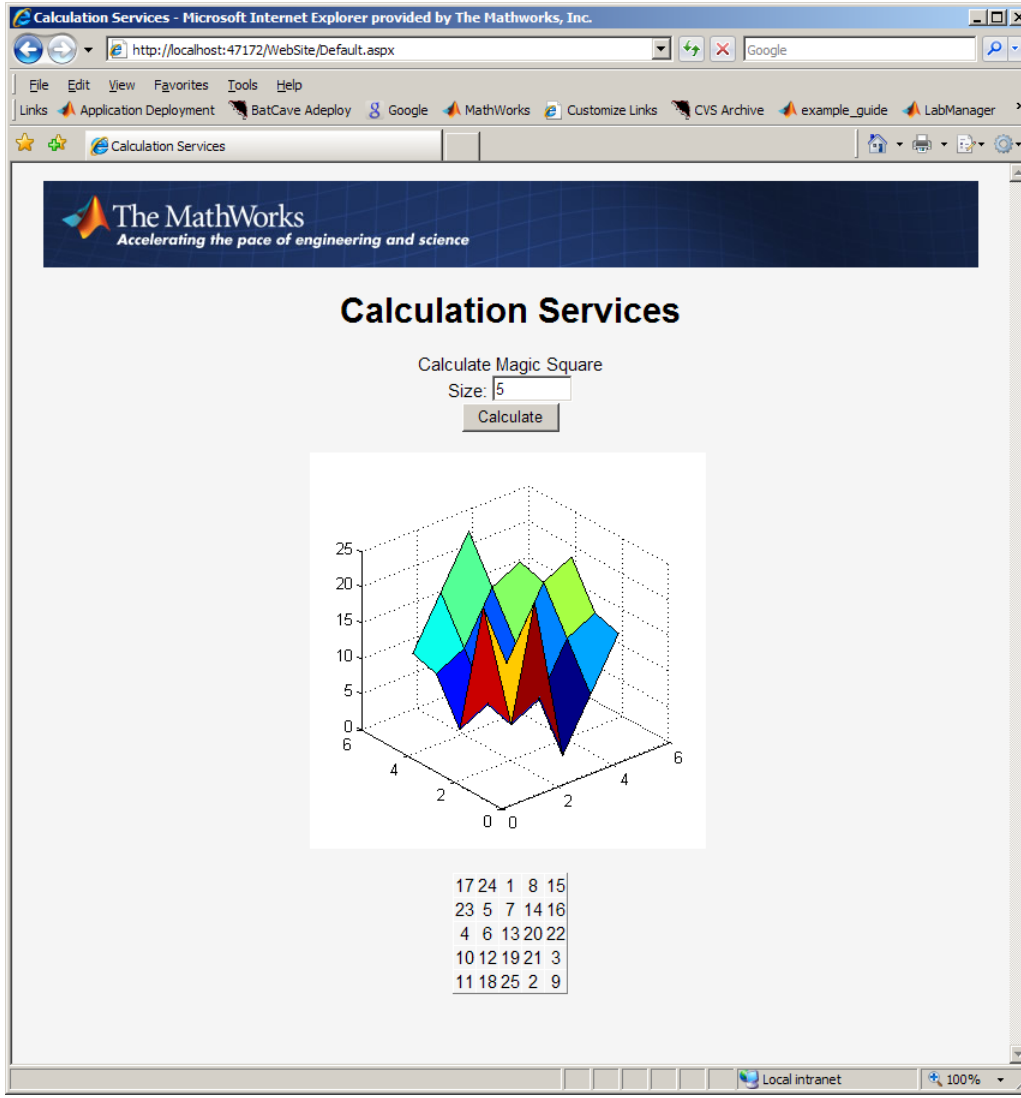
The `getMagicWebFigure.m` MATLAB function, which runs the Magic Square calculator, is as follows. It is based on the popular Magic Square `getMagic` function:**getmagic.m:**

```
function magicOutput = getMagic(x)
    magicOutput = magic(x);
end
```

getMagicWebFigure.m:

```
function figureOutput = getMagicWebFigure(x)
    f = figure;
    magicOutput = magic(x);
    surf(magicOutput);
    set(gcf,'Color',[1,1,1])
    figureOutput = webfigure(f);
    close(f);
end
```

The Magic Square Calculator application, when built, looks like this.



Magic Square Calculator Application Presented on the Web

Creating an End-to-End Web Application

In this section...

Creating a Java Web Application, End-to-End

In order to deploy the Magic Square Calculator application described in “Magic Square Calculator On the Web” on page 8-2 to the Web, you must build and run your application using Java or .NET.

To create the Magic Square Calculator in Java, you must create:

1. The JAR file generated by compiling MATLAB code using MATLAB Compiler SDK.
2. The JSP entry page, responsible for taking in user input, passing it on to the servlet, and displaying the servlet's result on the page.
3. The servlet, responsible for instantiating the deployed component. When a request comes in, calling the MATLAB function that returns the matrix, the MATLAB function returns the WebFigure. The servlet then binds the WebFigure to the application cache of the server and produces HTML that displays the WebFigure and finally, the matrix.

The following procedure gives you an option to build your example Web component or download it from MATLAB Central. To download the example, go to “Building Your Example Component” on page 8-4.

Building Your Example Component

1. Download the application code for this example from MATLAB Centrals File Exchange at `https://www.mathworks.com/matlabcentral/fileexchange`. Once you open the file exchange, search for “Java Web Example Guide End To End Chapter.”
2. Extract the `JavaEndToEnd.zip` file into a working folder where you can build the application.
3. Start Tomcat by changing your folder to `tomcat\bin` and executing `startup.bat`.
4. Copy `javabuilder.jar` to the `tomcat/common/libs` folder.

5 Once Tomcat starts successfully, drag the `JavaEndToEnd.war` file into the `webapps` folder under the `tomcat` folder.

6 Execute the application by opening a Web browser and pointing to `http://localhost:8080/JavaEndToEnd/MagicSquare/ExamplesPage.jsp`.

Building Your Example Component Manually

1 Ensure you have a version of the Java Developer's Kit (JDK™) installed that matches the version used by the MATLAB Runtime. See the MATLAB Compiler™ User's Guide reference pages for details on the `mcrversion` command.

2 Ensure you have Tomcat 5 or later on your system (other J2EE Web servers can work also, but the steps in this document have been tested with Tomcat).

3 Ensure the version of the MATLAB Runtime you have installed is the same version as the MATLAB Runtime running with MATLAB when the application was built. If you are unsure, check with your MATLAB programmer or whoever initially deployed the component.

4 Make note of the folder where the MATLAB Runtime is installed. It will be used later when starting the applications.

5 Create the code for the JSP page:

Note This code uses an image resource and a cascading style sheet resource that is included if you download the code from MATLAB Central as in “Building Your Example Component” on page 8-4.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
    <head>
        <title>Calculation Services</title>

        <%
            //This section of code determines if the user has entered
            // a number for the square size, if they have it overlays
            // the default size which is 5.
            String sizeStr = request.getParameter("size");
            int size = 5;
            if(sizeStr!=null && sizeStr.length()>0)
            {
                size = Integer.parseInt(sizeStr);
            }
        %>

        <link rel="Stylesheet" type="text/css"
                media=all href="./StyleSheet.css" />
```

```
    <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>

<body>
    <form method="get">
        <div style="text-align: center">
            <table width="760" cellpadding="0" cellspacing="0">
                <tr>
                    <td><img src="header_bg.jpg"
                              alt="Header Image Not Found"
                         width="779" height="72" /></td>
                </tr>
            </table>
            <br />

            <h1> Calculation Services</h1>

            Calculate Magic Square
            <br>
            Size:
            <input type="text" name="size" size="8" value="<%=size%>" >
            <br>
            <input type="submit" value="Calculate">
            <br>
            <br />
            <script type="text/javascript">
                try
                {
                    //Sets up an HttpRequest object so we can call our
                    // servlet and dump the output to the screen
                    var objXHR = new XMLHttpRequest();
                }
                catch (e)
                {
                    try
                    {
                        var objXHR = new ActiveXObject('Msxml2.XMLHTTP');
                    }
                    catch (e)
                    {
                        try
                        {
                            var objXHR =
                               new ActiveXObject('Microsoft.XMLHTTP');
                        }
                        catch (e)
                        {
                            document.write
                               ('XMLHttpRequest not supported');
                        }
                    }
                }
                //Call the MagicSquare Servlet and pass it the
                //          size of the matrix to show
                objXHR.open('GET','MagicSquare?size=<%=size%>',false);
```

```
                    objXHR.send(null);

                    //Display the result of the servlet on the page
                    document.writeln(objXHR.responseText);
                </script>
                <br>
            </div>
        </form>
    </body>
</html>
```

6 Create the code for the servlet:

Note This code requires that the generated component created earlier and `javabuilder.jar` must be on the `classpath` in order to compile.

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
import javax.servlet.ServletConfig;
import java.io.IOException;
import examples.*;
import com.mathworks.toolbox.javabuilder.webfigures.WebFigure;
import com.mathworks.toolbox.javabuilder.webfigures.WebFigureHtmlGenerator;
import com.mathworks.toolbox.javabuilder.MWJavaObjectRef;
import com.mathworks.toolbox.javabuilder.MWNumericArray;
import com.mathworks.toolbox.javabuilder.MWException;

public class MagicSquareServlet extends HttpServlet
{
    private MagicCalc calc;
    public void init(ServletConfig config) throws ServletException
    {
        super.init(config);

        try
        {
            //We initialize the deployed component in the init
            // method so it doesn't get initialized with each request
            calc = new MagicCalc();
        }
        catch(MWException e)
        {
            e.printStackTrace();
        }
    }

    public void destroy()
    {
        super.destroy();
```

```
        if(calc!=null)
        {
            //When the servlet gets disposed you can clean
            // up the deployed component reference as well.
            calc.dispose();
        }
    }

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException
    {
        //If no size has been provided, return.
        String sizeStr = request.getParameter("size");
        if(sizeStr==null || sizeStr.length()==0)
            return;

        //Convert the input parameter size to an MWArray
        //    for use with our components
        MWNumericArray size = new MWNumericArray(Integer.parseInt(sizeStr));
        double[][] square = new double[0][];
        WebFigure figure = null;
        try
        {
//Call the getMagicWebFigure method and turn
            // the java object reference into a WebFigure object
            Object[] results = calc.getMagicWebFigure(1, size);
            MWJavaObjectRef ref = (MWJavaObjectRef)results[0];
            figure = (WebFigure)ref.get();
            //Attach the WebFigure to the Servlets Application cache
            getServletContext().setAttribute("UserPlot",figure);


            //Call the getMagic method and turn the
            //    MWArray into a array of doubles
            Object[] result = calc.getMagic(1, size);
            MWNumericArray array = (MWNumericArray)result[0];
            square = (double[][])array.toArray();
        }
        catch(MWException e)
        {
            e.printStackTrace();
        }

        StringBuffer buffer = new StringBuffer();

        //The WebFigureHtmlGenerator class creates an HTML string that
        // can be embedded into a web page and will create an iFrame
        // that contains the WebFigure.
        WebFigureHtmlGenerator webFigureHtmlGen =
                new WebFigureHtmlGenerator("WebFigures",getServletContext());

        if(figure!=null)
        {
            try
```

```
            {
                String outputString =
                        webFigureHtmlGen.getFigureEmbedString(
                                figure,
                                "UserPlot",
                                "application",
                                "330",
                                "330",
                                null);
                buffer.append(outputString);
            }
            catch(Exception e)
            {
                e.printStackTrace();
            }
        }

        buffer.append("<BR>");
        buffer.append("<BR>");

        //Step through the matrix and output it to an HTML Table
        buffer.append("<TABLE >");
        for (double[] row : square)
        {
            buffer.append("<TR>");
            for (double value : row)
            {
                buffer.append("<TH>");
                buffer.append(new Double(value).intValue());
            }
        }
        buffer.append("</TABLE>");
        buffer.append("<BR>");


        //Write the embeddable html to the response
        // output stream for use on the jsp page
        response.getOutputStream().print(buffer.toString());
    }
}
```

7. Copy the `javabuilder.jar` file (from *`matlabroot`*`/toolbox/javabuilder/jar/javabuilder.jar`) to the `webapp/WEB-INF/lib` folder.
8. Since you are compiling a servlet which has J2EE dependencies, copy `servlet-api.jar` (the J2EE JAR file that comes with Tomcat) to the current working folder. This file is usually located in the Tomcat common `lib` folder (a J2EE JAR file can work as well).
9. From the root of your `Projects` folder we need to build the component with MATLAB by using the following `mcc` command:

```
mcc -W "java:examples,MagicCalc" -d .\scratch -T "link:lib"
    -v "class{MagicCalc:.\webapp\WEB-INF\mcode\getMagic.m,
        .\webapp\WEB-INF\mcode\getMagicWebFigure.m}"
```

10 Copy it from your working folder to the Web applications `lib` folder by typing:

```
xcopy /Y .\scratch\examples.jar .\webapp\WEB-INF\lib
```

11 Rebuild the servlet by typing the following command:

```
javac -cp servlet-api.jar;
                      .\webapp\WEB-INF\lib\javabuilder.jar;
                      .\webapp\WEB-INF\lib\examples.jar;
       -d .\webapp\WEB-INF\classes
          .\webapp\WEB-INF\src\MagicSquareServlet.java
```

Note This assumes JDK 1.6 `javac` is on your system path:

12 To rebuild the WAR file, enter:

```
cd webapp
jar -cvf ..\JavaEndToEnd.war .
cd ..
```

Creating a .NET Web Application, End-to-End

To create the Magic Square Calculator using .NET, you must create:

1 The MATLAB Compiler SDK DLL file that is generated by compiling MATLAB code using MATLAB Compiler SDK.

2 The `DataTable` implementation that converts the magic square output into something that an `ObjectDataSource` can use in a `GridView` Control.

3 The ASPX page and the code behind it. This page is responsible for taking in user input, displaying the controls, and handling page events.

Preparing Your Example Application

1 Ensure you have a supported .NET Framework installed and Microsoft Visual Studio. See the MATLAB Compiler SDK system requirements for a list of supported framework versions.

2 Ensure the version of the MATLAB Runtime you have installed is the same version as the MATLAB Runtime running with MATLAB when the application was built. If you

are unsure, check with your MATLAB programmer or whoever initially deployed the components.

3. Make note of the folder where the MATLAB Runtime is installed. It will be used later when starting the applications.

4. Create the code for the `DataTable` implementation:

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using Examples;
using MathWorks.MATLAB.NET.Arrays;

public class MagicSquare
{
    //This class gets the data from the generated component and loads
    //  it into a data structure that can be automatically used by an
    //  ObjectDataSource in a GridView Control
    public DataTable getMagicSquare(int size)
    {
        //Instantiate the deployed component
        MagicCalc calc = new MagicCalc();

        //Convert the size into an MWArray so it can be passed in
        MWArray input = size;

        //call the getMagic function to get the matrix
        MWNumericArray output = (MWNumericArray)calc.getMagic(input);

        //Convert the matrix to a .NET double array
        double[,] magicSquare =
            (double[,])output.ToArray(MWArrayComponent.Real);

        //Create an empty data table to put the matrix data in.
        DataTable table = new DataTable();

        //Since we know its a square add as many
        //  columns as there will be rows.
        for(int i = 0; i<size; i++)
        {
            table.Columns.Add();
        }

        DataRow row;
        //Iterate each element in the array creating a row out of each
        for(int i = 0; i < size;i++)
        {
            //create a row from the table to put the data in
```

```
            row = table.NewRow();
            //Iterate each element in the inner
            //  array and put them into the row
            for (int j = 0; j < size; j++)
            {
                row[j] = magicSquare[i,j];
            }

            //Add the row to the table
            table.Rows.Add(row);
        }
        return table;
    }
}
```

5 Create the code for the ASPX page:

```
<%@ Page Language="C#" AutoEventWireup="true"
     CodeFile="Default.aspx.cs" Inherits="_Default" %>

<%@ Register
Assembly="WebFiguresService,
               Version=2.8.1.0, Culture=neutral,
    PublicKeyToken=e1d84a0da19db86f"
        Namespace="MathWorks.MATLAB.NET.WebFigures.Service"
TagPrefix="cc1" %>

<!DOCTYPE html
 PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
    <head id="Head1" runat="server">
        <title>Calculation Services</title>
        <link rel="Stylesheet" type="text/css"
                    media=all href="./StyleSheet.css" />
        <link href="StyleSheet.css"
                    rel="stylesheet" type="text/css" />
    </head>
    <body>
        <form id="form1" runat="server">
 <div style="text-align: center">
                <table width="760" cellpadding="0"
                                    cellspacing="0">
                    <tr>
                        <td>
<img
src="header_bg.jpg"
```

```
alt="Header Image Not Found"
width="779"
height="72" /></td>
                    </tr>
                </table>
                <br />

                <h1> Calculation Services</h1>

                <asp:Label
ID="Label3"
runat="server"
Text="Calculate Magic Square">
</asp:Label>
<br />
               <asp:Label
ID="Label4"
runat="server"
Text="Size: ">
</asp:Label>

<asp:TextBox
ID="TextBox3"
runat="server"
Width="61px">
5
</asp:TextBox>
<br />
               <asp:Button
ID="Button2"
runat="server"
OnClick="Button2_Click"
Text="Calculate" />
<br />

                <br />

                <cc1:WebFigureControl
ID="WebFigureControl1"
runat="server"
Height="330px"
Width="330px" />
                <br />
```

```
                    <br />

                    <asp:ObjectDataSource
                        ID="ObjectDataSource1"
                        runat="server"
                        SelectMethod="getMagicSquare"
                        TypeName="MagicSquare">

                        <SelectParameters>
                            <asp:ControlParameter
                                ControlID="TextBox3"
                                DefaultValue="5"
                                Name="size"
                                PropertyName="Text"
                                Type="Int32" />
                        </SelectParameters>
                    </asp:ObjectDataSource>
                    <asp:GridView
                        ID="GridView1"
                        runat="server"
                        DataSourceID="ObjectDataSource1"
                        ShowHeader="False">
                    </asp:GridView>
                </div>
            </form>
        </body>
    </html>
```

6 Create the code behind the ASPX page:

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using Examples;
using MathWorks.MATLAB.NET.WebFigures;
using MathWorks.MATLAB.NET.Arrays;

public partial class _Default : System.Web.UI.Page
{
    private MagicCalc calc = new MagicCalc();
    protected void Page_Load(object sender, EventArgs e)
    {
        MWArray input = Int32.Parse(TextBox3.Text);
```

```
            WebFigureControl1.WebFigure =
                new WebFigure(calc.getMagicWebFigure(input));
        }
        protected void  Button2_Click(object sender, EventArgs e)
        {
            MWArray input = Int32.Parse(TextBox3.Text);
            WebFigureControl1.WebFigure =
                new WebFigure(calc.getMagicWebFigure(input));

            ObjectDataSource1.Select();
        }
    }
```

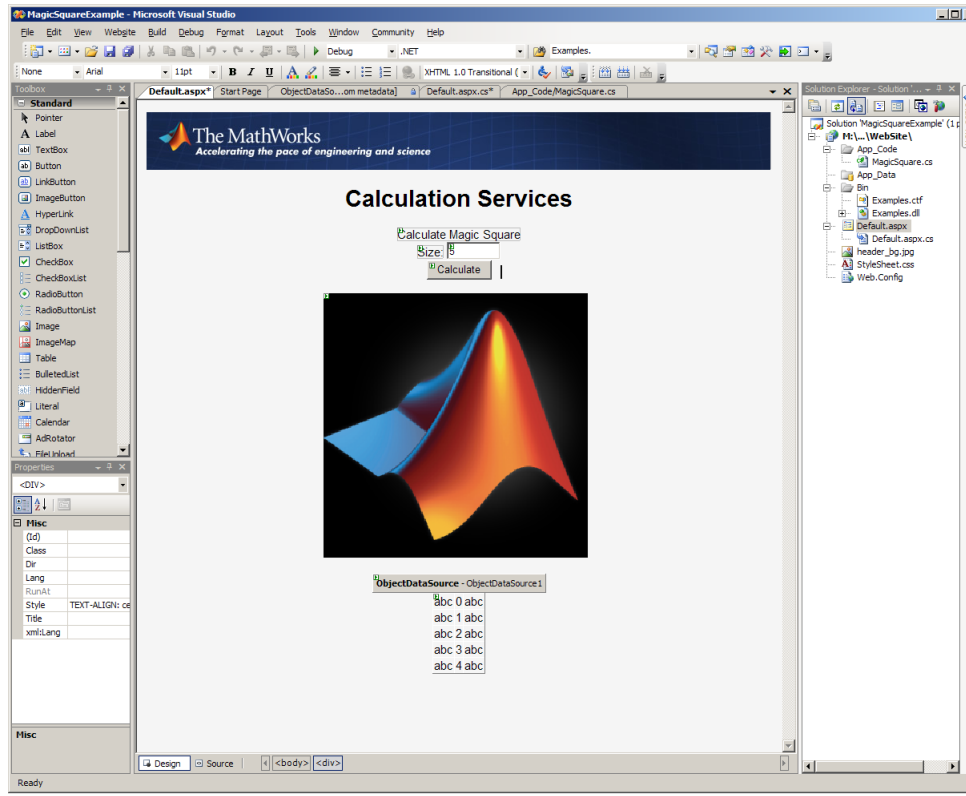
Building Your Example Application

Build your application and deploy it using the version of IIS built into Microsoft Visual Studio as follows:

1. From the root of your `projects` folder, open the `WebSite` folder.
2. Build the component in MATLAB using the `mcc` command:

   ```
   mcc -W "dotnet:Examples,MagicCalc,0.0,private"
       -d ".\mcode\output" -T "link:lib"
       -v
        "class{MagicCalc:.\mcode\getMagic.m,
               .\mcode\getMagicWebFigure.m}"
   ```

   This command creates a file in the `WebSite\mcode\output` folder called `Examples.dll` (this will be referenced from the Visual Studio project in the next step).
3. Open Microsoft Visual Studio and select **File > Open > Web Site**.
4. Browse to the `WebSite` folder and open it.
5. Confirm that `Examples.dll` is added correctly as a reference. If there were any errors or issues adding it, delete it and add a new reference to the DLL you built in the `WebSite\mcode\output` folder.



Microsoft Visual Studio Designer View of the Magic Square Calculator

6 Select **Build > Build Solution**. You should build with no errors. If there are errors, ensure that:

- The MATLAB Runtime is installed.
- `Examples.dll` has been added as a reference in your project.

7 When you have built successfully, select **Debug > Start Debugging**. A local IIS server starts and opens your page inside of it.