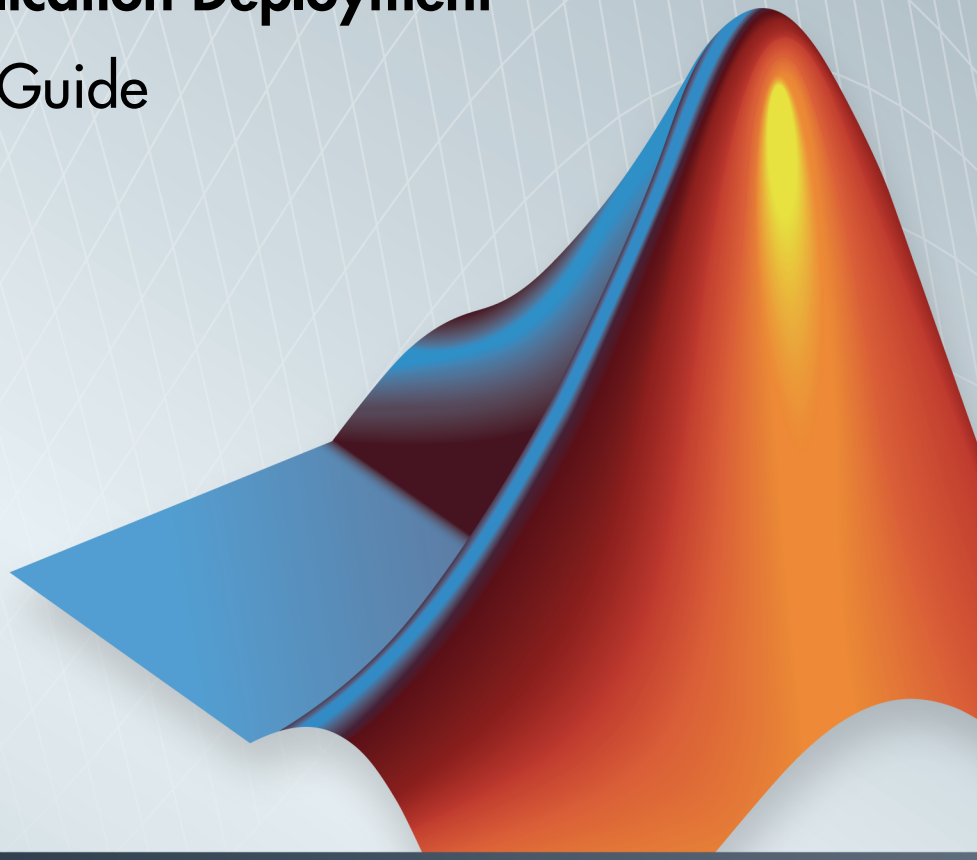
MATLAB® Application Deployment

Web Example Guide

R2014**b**



MATLAB®

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

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

Phone: 508-647-7000

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

MATLAB® Application Deployment Web Example Guide

Revision History

March 2009	Online only	New for Version 2.0.3 (Release R2009a)
September 2009	Online only	Revised for Version 2.0.4 (Release R2009b)
March 2010	Online only	Revised for Version 2.1 (Release R2010a)
September 2010	Online only	Revised for Version 2.2 (Release R2010b)
January 2011	Online only	Revised for Version 2.2.1 (Release R2010bSP1)
April 2011	Online only	Revised for Version 2.2.2 (Release R2011a)
September 2011	Online only	Revised for Version 2.2.3 (Release R2011b)
March 2012	Online only	Revised for Version 2.2.4 (Release R2012a)
September 2012	Online only	Revised for Version 2.2.5 (Release R2012b)
March 2013	Online only	Revised for Version 2.2.6 (Release R2013a)
September 2013	Online only	Revised for Version 2.3 (Release R2013b)
March 2014	Online only	Revised for Version 2.3.1 (Release R2014a)
October 2014	Online only	Revised for Version 2.3.2 (Release R2014b)

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

A Sources for More Information

1

How to Use This Guide

The MATLAB Application Deployment Web Example Guide

MATLAB® Compiler™, MATLAB Builder NE™, MATLAB Builder JA™, and MATLAB Builder EX™ take MATLAB functions and expose them in a language-specific manner that can be deployed to users who do not have MATLAB installed.

The goal of the *MATLAB Application Deployment Web Example Guide* is to provide a series of templates demonstrating how to successfully implement the possible configurations available in the Web deployment space.

Access the guide from the MathWorks® Web site at:

- MATLAB® Compiler™ — http://www.mathworks.com/help/pdf_doc/compiler/index.html.
- MATLAB Builder™ JA — http://www.mathworks.com/help/pdf_doc/javabuilder/index.html
- MATLAB Builder NE — http://www.mathworks.com/help/pdf_doc/dotnetbuilder/index.html

Use the *MATLAB Application Deployment 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 the most common to the most complex deployment tasks, such as:
  - Creating a deployable function
  - Hosting the component delivered by the MATLAB Programmer using J2EE and .NET Web technologies
  - Displaying complex data types (arrays, matrices) on a Web page
  - Enabling scalability through stateless services used with MATLAB Builder JA or MATLAB Builder NE
- Deploying applications through implementation of SOAP Web services

Who Should Use This Guide?

Many skill sets are involved in deploying MATLAB® applications.

These skills sets include the MATLAB Programmer (usually a scientist or engineer), a Business Services Developer and front-end Web developer (programmers responsible for interfacing with languages and frameworks such as Java and .NET, as well as developing Web page content), and the end user, who consumes the final product.

Since it is sometimes confusing to determine who should perform what task in a large installation, this guide's structure is role based. In other words, the tasks that belong to each role are listed in separate sections or chapter. This organization enables more novice users to focus only on the tasks related to their area of expertise and enables advanced users to customize a list of tasks pertinent to their own area of expertise.

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 2-13.

Commonly Used Software and Preliminary Setup Information

In this section...

MATLAB Programmer

MATLAB Programmer

Role	Knowledge Base	Responsibilities
MATLAB programmer	• MATLAB expert • No IT experience • No access to IT systems	• Develops models; implements in MATLAB • Uses tools to create a component that is used by the Java developer

- MATLAB®
- Financial Toolbox
- MATLAB Compiler
- MATLAB Builder JA
- MATLAB Builder NE

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)

Middle-Tier Developer

Middle-tier developer	• Little to no MATLAB experience • Moderate IT Experience • Expert at business logic and services tier	• Integrates deployed component with the rest of the J2EE system by converting MATLAB data types to the Java Business logic objects

	• Java® expert • Minimal access to IT systems • Expert at J2EE • Expert at Web services	

Front-End Web Developer

Role	Knowledge Base	Responsibilities
Front-end Web developer	• No MATLAB experience • Minimal IT experience • Expert at usability and Web page design • Minimal access to IT systems • Expert at JSP	• As service consumer, manages presentation and usability • Creates front-end applications • Integrates MATLAB code with language-specific frameworks and environments • Integrates WebFigures with the rest of the Web page

- 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

End User

Role	Knowledge Base	Responsibilities
End user	• No MATLAB experience • Some knowledge of the data that is being displayed, but not how it was created	• In Web environments, consumes what the front-end developer creates • Integrates MATLAB code with other third-party applications, such as Excel

- 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...

Web Processing Work Flow

Web processing occurs between a client that makes requests and a Web server that responds to the requests. The communication between client and server is done using HTTP (Hypertext Transfer Protocol). HTTP is a simple, synchronous protocol where a client connects to a server, sends a single command, and waits for a response.

Client requests consist of two items:

- an HTTP verb such as `GET`
- a URL

  URLs contain two pieces of information:

  - the address of the server
  - the object of the HTTP verb

The exchange follows the pattern:

1. The client constructs a request.

   One common scenario is typing a URL into the address bar of your Web browser. The browser adds a `GET` to the URL to create the full request.
2. The client sends the request over the wire using HTTP.
3. The client begins to wait for a response.
4. The server receives the request.
5. The server acknowledges the request by sending a message to the client.

6 The client continues to wait for the request to be fulfilled.
7 Based on the HTTP verb in the request, the server processes the request.

   In the case of a Web browser, the verb is typically GET. The server retrieves, or constructs, the requested HTML page.
8 The server sends a response to the client.
9 The client processes the response.

   In the case of a Web browser, the client renders the returned HTML page.

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.

One of the simplest ways to integrate MATLAB code in a Web context is to write a Java servlet, and in its `doGet` method, make calls against an object built by MATLAB Builder JA.

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`s 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 (server).

For an example of how to use JSP pages to dynamically render WebFigures, see "Creating a WebFigure on a JSP Page" on page 5-6 and "Creating an End-to-End Web Application" on page 8-5 in this Example Guide, and "Implement a Custom WebFigure" in the User's Guide.

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 an 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`, 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 2-5, `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 2-8.

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);
```

The Oracle Web Site: Your Ultimate Java Resource

For the most detailed coverage of the Java language on the Web, refer to Oracle's Web Site at http://www.oracle.com/us/technologies/java/index.htm.

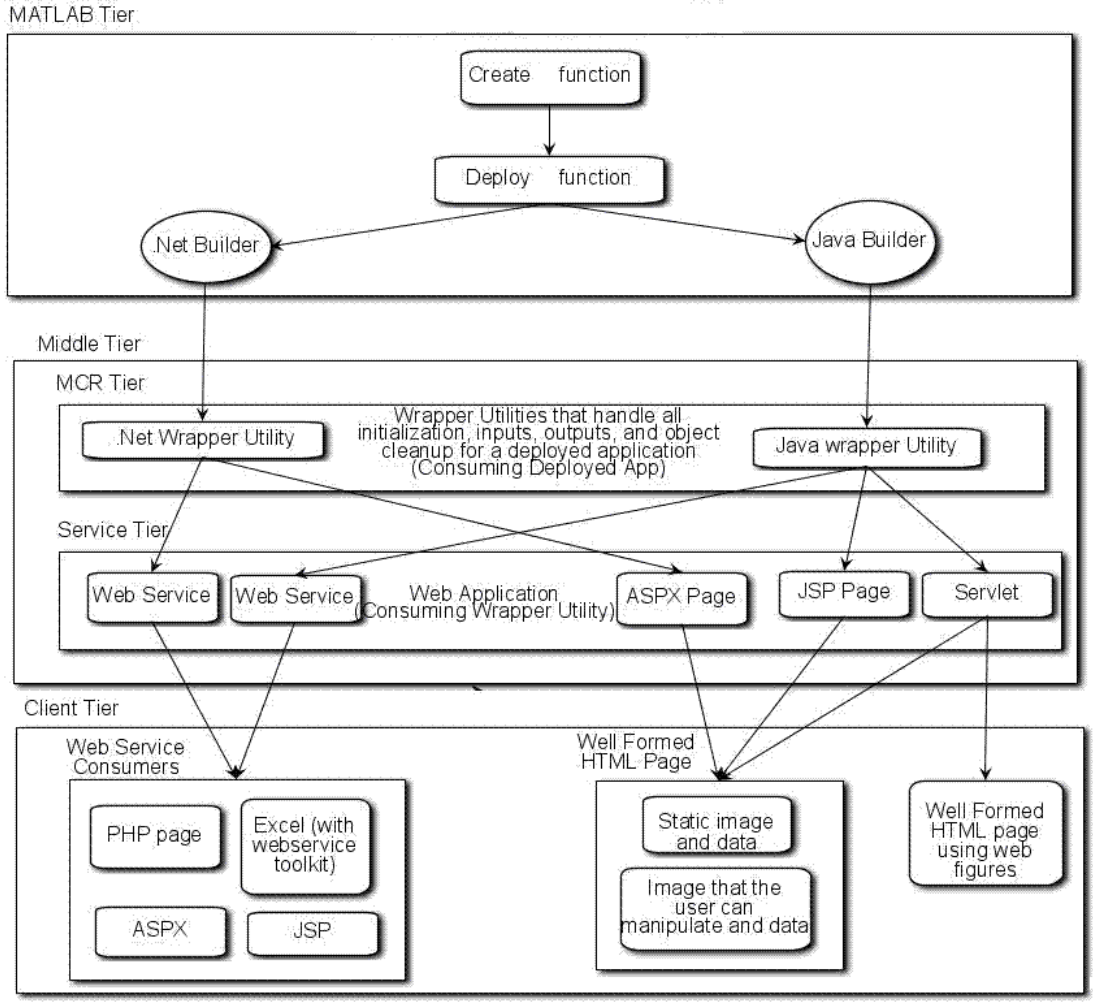
Lifecycle of a Deployed MATLAB Web Application

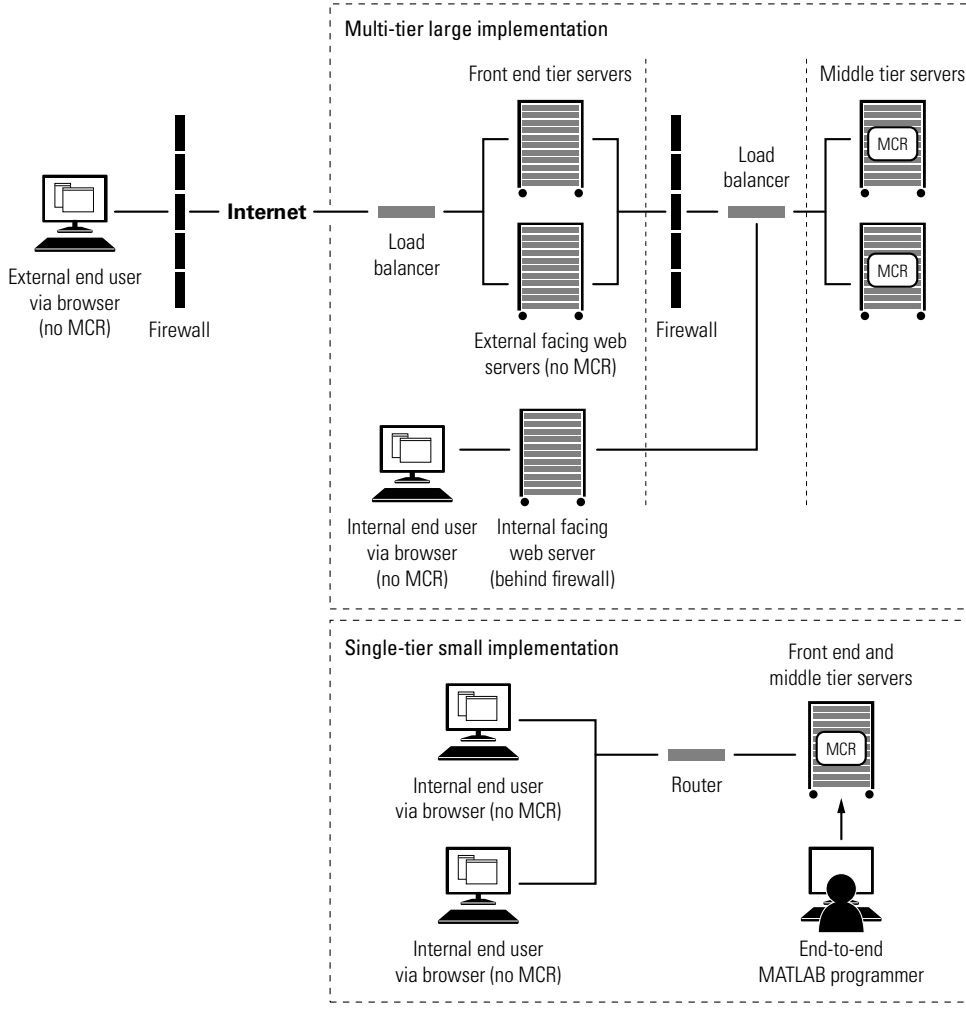
In this section...

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
.Net Builder
Java Builder
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 Application Deployment products (MATLAB Compiler and the builders) is to enable work that has been accomplished within MATLAB 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 builder products allow integration of the 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 Builder JA and MATLAB Builder NE 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, whose objective is sharing it with other people either within or outside of the organization. To accomplish this objective, the programmer uses MATLAB Compiler. MATLAB Compiler makes MATLAB code usable by people in vastly different environments who may not have knowledge of MATLAB or the MATLAB language.

When MATLAB Builder JA (for Java language) or MATLAB Builder NE (for Microsoft .NET Framework) is installed along with MATLAB Compiler, MATLAB functions can be encrypted and wrapped in Java or .NET interfaces. The MATLAB programmer takes these deployable components and gives them to the middle-tier developer.

Middle-Tier Developer

At this point in the deployment lifecycle, integration is usually required in order to make the deployed application work with the existing applications in the organization. The Business Services Developer installs these deployable applications along with the proper version of the MATLAB runtime, and converts MATLAB data types to native language data types so they can be used without any coupling to MATLAB in other tiers of the installation architecture. When the Java or .NET component is called, it will instantiate the runtime to execute the underlying MATLAB code. Once these services are exposed (either as Web services or through an API) Front End Developers can connect to them and use them.

Front-End Web Developer

Front-end Web 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 front-end Web 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. End users typically achieve this through a Web browser.

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

The end-to-end developers are virtual “one-stop shops.” They are MATLAB experts, but are also skilled in many of the areas of expertise as the middle-tier developer and front-end Web developer, though their level of expertise may vary over their many areas of responsibility. 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

In this section...

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 a MATLAB function to other programming environments, such as .NET and Java, you must contain your MATLAB code within functions. MATLAB Compiler does not allow you to use inline scripts.

The following examples demonstrate how to perform basic MATLAB programmer tasks for deployed applications; they do not attempt to represent every way a MATLAB programmer can interface with MATLAB. Upcoming topics demonstrate how to use various data types in deployed applications. For more specific information about any of these data types, see your product User's Guide.

MATLAB Programming Basics

Creating a Deployable MATLAB Function

Virtually any calculation that you can create in MATLAB can be deployed, if it resides in a function. For example:

```
>> 1 + 1
```

cannot be deployed.

However, the following calculation:

```
function result = addSomeNumbers()
 result = 1+1;
end
```

can be deployed because the calculation now resides in a function.

Taking Inputs into a Function

You typically pass inputs to a function. You can use primitive data type as an input into a function.

To pass inputs, put them in parentheses. For example:

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

Returning MATLAB Data Types

MATLAB allows many different deployable data types. This section contains examples of how to work with figures. For an in-depth explanation of how to work with MATLAB primitive data types, see the MATLAB External Interfaces documentation.

MATLAB Figures

Often, you are dealing with images displayed in a figure window, and not just string and numerical data. Deployed Web applications can support figure window data in a number of ways. By using the WebFigures infrastructure (see “About the WebFigures Feature” in the MATLAB Builder JA documentation or “WebFigures” in the MATLAB Builder NE documentation), the respective builder marshals the data for you.

Alternatively, you can take a snapshot of what is in the figure window at a given point and convert that data into the raw image data for a specific image type. This is particularly useful for streaming the images across the web.

Returning Data from a WebFigure Window

WebFigures is a feature that enables you to embed dynamic MATLAB figures onto a Web page through a Builder JA or Builder NE component. This concept can be used with any data in a figure window.

As in the following example, you close the figure before the code is exited so that the figure does not “pop up,” or appear later, in the deployed application. You do not need to specify any reorientation data when using WebFigures. If the figure is attached to the rest of the infrastructure, it will automatically pass, resize, and reorient accordingly.

```
%returns a WebFigure reference containing the
%data from the figure window
function resultWebFigure = getWebFigure
   f = figure;
   set(f,'Color',[.8,.9,1]);
   f = figure('Visible','off');
   surf(peaks);
   resultWebFigure = webfigure(f);
   close(f);
end
```

Returning a Figure as Data

This approach is typically used for instances where WebFigures can't be used, or in a stateless application.

```
%We set the figure not to be visible since we are
%streaming the data out
%Notice how you can specify the format of the bytes,
%  .net uses unsigned bytes (uint8)
%  java uses signed bytes (int 8)
%This function allows you to specify the image format
%such as png, or jpg
function imageByteData = getSurfPeaksImageData(imageFormat)
  f = figure;
  surf(peaks);
  set(f, 'Visible', 'off');
  imageByteData = figToImStream(f, imageFormat, 'uint8');
  close(f);
end
```

Reorienting a Figure and Returning It as Data

Sometimes you want the function to change the perspective on an image before returning it. This can be accomplished like this:

```
%We set the figure not to be visible since we are
%streaming the data out
%Notice how you can specify the format of the bytes,
%  .net uses unsigned bytes (uint8)
%  java uses signed bytes (int 8)
```

```
%This function allows you to specify the image format
%such as png, or jpg
function imageData =
     getImageDataOrientation(width, height, rotation,
                                 elevation, imageFormat)
  f = figure('Position', [0, 0, width, height]);
  surf(peaks);
  view([rotation, elevation]);
  set(f, 'Visible', 'off');
  imageData = figToImStream (f, imageFormat, 'uint8');
  close(f);
end
```

Deploying MATLAB Code with the Builders

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.

For examples using the MATLAB Library Compiler, see “Implement a Custom WebFigure” in the MATLAB Builder JA documentation and “WebFigures” in the MATLAB Builder NE documentation.

It is also possible to use the MATLAB® Compiler™ `mcc` command to build components.

Middle-Tier Developer Tasks

Working with the Business Service Layer

In this section...

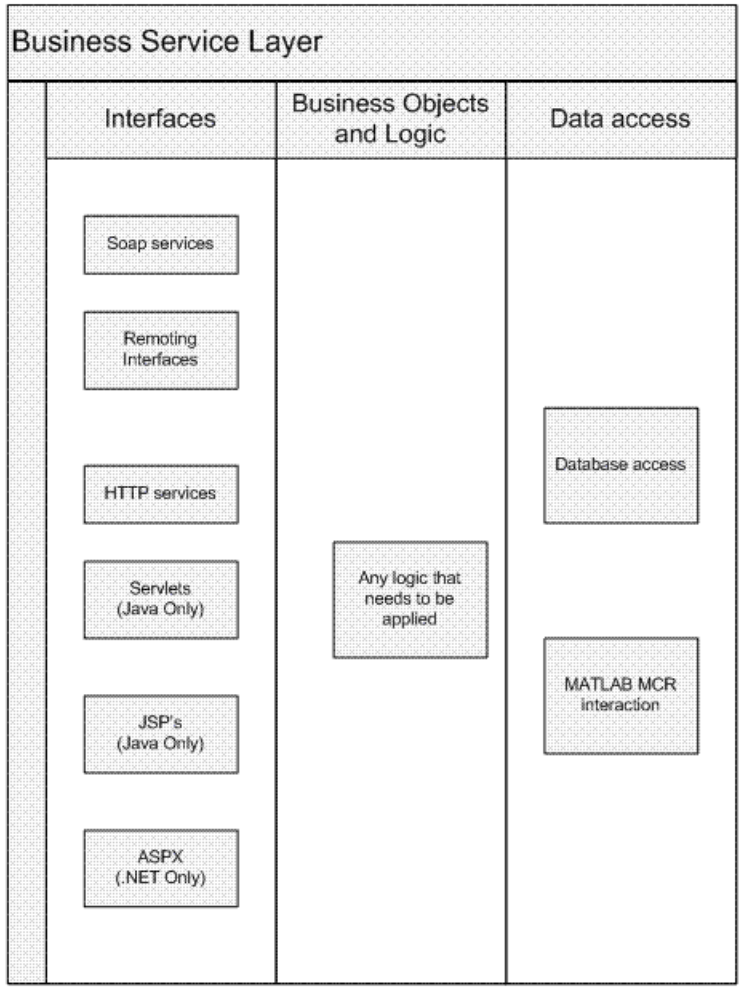
About the Business Service Layer

Note: For comprehensive end-to-end implementations of the concepts in this chapter, see Appendix A.

Most software contains a *business service layer*, which is a set of interfaces, business objects (and logic to manipulate them), and mechanisms for data access that run the core business.

A typical business service layer contains the following sublayers:

- Interfaces — Typically the business service layer can implement several different interface types which all interact with common data elements and common business objects, all using the same business logic. Software and related services used to access business data from native or Web clients include:
  - SOAP services
  - Remoting interfaces
  - HTTP services
  - Java servlets
  - JSPs (for Java)
  - ASPX (for .NET)
- Business Objects and Logic — This is business data expressed in the form of objects along with the logic to manipulate the objects. This data is loaded by a combination of inputs from the interfaces and data from the data access layer.
- Data Access — This layer links to all lower-level data such as databases, where access into a deployed application would typically take place. Your generated component fits into this category, as it can be used as a mechanism through which to access the MATLAB runtime.



Elements of the Business Service Layer

About the Examples

Depending on the size and complexity of an implementation some of these elements can overlap. The examples in this documentation assume direct communication from the interfaces into the DAO (data access object or wrapper utility) that you create.

All examples in this document are coded as stateless (with the exception of the MATLAB Builder JA™ WebFigures example) and are scalable. Servers can be added or augmented by a load balancer for performance tuning.

Tip To scale MATLAB Builder NE WebFigures, use the .NET Distributed Session setting. This enables all machines in your Web farm to use the same session.

Creating a DAO 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 builder products 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();
}
```

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 4-5and the same component class is used. The Java and .NE Builder infrastructure handles data marshaling when passing parameters to a component. Data conversion rules can be found in the MATLAB builder 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);
```

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]);
}
```

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);
}
```

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));
}
```

Getting MATLAB Numerics from a Deployed Application

This code resides in the `try` block for an initialized component (see "Initializing a Component" on page 4-5).

Java

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

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);
```

```
}
```

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);
}
```

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);
}
```

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);
}
```

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);
}
```

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);
}
```

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

Java

For more information about WebFigures, see “About the WebFigures Feature” in the MATLAB Builder JA documentation.

```
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.

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);
    }
}
```

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();
    }
}
```

Hosting a DAO on a Web Server

After you construct your DAO, you need to expose the wrapped service(s) via the Web.

There are many things to consider with regards to exposing the service. For example, a JSP is not suited for binary streaming since the J2EE infrastructure already wraps the output stream. In each of the following sections, some basic concepts that can be used in a realistic system are demonstrated. Typically, the response is not simply dumped to the output stream, but instead wrapped in a more complex XML document or Web service. Using these templates as a guide, you can extend these examples using similar patterns. For each of these examples, refer to the DAO class defined in "Creating a DAO for Deployment" on page 4-5. This DAO takes care of MATLAB specific data conversion and data clean-up tasks.

Hosting the DAO with a Servlet

Note that the DAO is initialized in the `init` method of the servlet. When you create and access a component created with the builders, an instance of the MATLAB runtime is created that the component communicates with in order to handle MATLAB tasks. This can incur much overhead if performed every time a user accesses the servlet. Alternately, by performing initialization in the `init` method, it is performed once for all sessions using the servlet. If you want to rebuild each time, place the call within a `doget` method.

It is also possible that neither of the above approaches will meet your needs since they initialize once per servlet, rather then once per server. If this is an issue, use a singleton object that is instantiated in a Context Listener class (a class that extends `ServletContextListener`). This class has a `contextInitialized` method and a `contextDestroyed` method which get called automatically when the server starts or is stopped. This allows all of your applications to access the singleton and access component objects as needed.

1. Create a staging folder, if one does not exist, under the folder where your component resides on your Web server. The DAO must reside in this folder, in a Java archive file (JAR), on the `class` path so it can be imported.
2. Initialize the DAO using the following examples as templates:

Initializing the DAO for a Servlet

```
package examples;
```

```
import java.io.IOException;
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.util.*;
import com.mathworks.toolbox.javabuilder.webfigures.WebFigure;

public class ExamplesServlet extends HttpServlet
{
  Examples examples = null;

  public void init(ServletConfig config) throws
                                  ServletException
  {
    super.init(config);

    try
    {
      examples = new Examples();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
  }

  public void destroy()
  {
    super.destroy();
    examples.dispose();
  }

  protected void doGet(final HttpServletRequest request,
HttpServletResponse response)
  throws ServletException, IOException
  {
    try
    {
      //*********************************
      //**All code using the DAO would go here
      //**Any of the below examples can be pasted here
```

```
      //**********************************
      int integer = examples.getIntFromMWNumericArray();
      response.getOutputStream().println("int: " + integer);
    }
    catch(Exception e)
    {
      e.printStackTrace();
      response.getOutputStream().println("ERROR "+
                                          e.getMessage());
    }
  }
}
```

Numeric

```
int integer = examples.getIntFromMWNumericArray();
response.getOutputStream().println("int: " + integer);
```

String

```
String string = examples.getStringFromMWCharArray();
response.getOutputStream().println("String: " + string);
```

Numeric Array

```
int[] intArray = examples.getIntArrayFromMWNumericArray();
response.getOutputStream().println("Numeric Array: ");
for(int i = 0; i<intArray.length;i++)
{
  response.getOutputStream().println("Array index("+ i+"): " +
                                     intArray[i]);
}
```

Character Array

```
char[] charArray = examples.getCharArrayFromMWCharArray();
response.getOutputStream().println("Char Array: ");
```

```
for(int i = 0; i<charArray.length;i++)
{
  response.getOutputStream().println
                 ("Array index("+ i +"): " +
                                charArray[i]);
}
```

Cell Array To Array

```
Object[] array = examples.getArrayFromCellArray();
for(int i = 0; i < array.length; i++)
{
  response.getOutputStream().println("Array index("+ i+"): " +
                                   array[i]);
}
```

Cell Array To List

```
List list = examples.getListFromCellArray();
Iterator listItr = list.iterator();
while(listItr.hasNext())
{
  response.getOutputStream().println("List Item: " +
                                          listItr.next());
}
```

Structure To Map

```
Map map = examples.getMapFromStruct();
response.getOutputStream().println("Structure Array: ");
Iterator mapKeyItr = map.keySet().iterator();
while(mapKeyItr.hasNext())
{
 String mapKey = (String)mapKeyItr.next();
 Object mapValue = map.get(mapKey);
 response.getOutputStream().println("KEY: " + mapKey + "   " +
    "VALUE: " + mapValue);
}
```

Byte Array

```
byte[] byteArray = examples.getByteArrayFromMWNumeric();
response.getOutputStream().println("Byte Array: ");
for(int i = 0; i<byteArray.length;i++)
{
  response.getOutputStream().print(byteArray[i]);
}
response.getOutputStream().write(byteArray);
```

Images (WebFigures)

This example is very similar to examples in the MATLAB Builder JA documentation, but this example guide code also uses our DAO.

```
HttpSession session = request.getSession();

WebFigure userPlot =
                (WebFigure)session.getAttribute("UserPlot");

// if this is the first time doGet has been called
//  for this session,
//  create the plot and WebFigure object
if (userPlot== null)
{
  userPlot = examples.getWebFigureFromMWJavaObjectRef();
  // store the figure in the session context
  session.setAttribute("UserPlot", userPlot);

 // bind the figure's lifetime to the session
  session.setAttribute(
    "UserPlotBinder",
    new MWHttpSessionBinder(userPlot));
}

WebFigureHtmlGenerator webFigureHtmlGen =
 new WebFigureHtmlGenerator(“WebFigures”,getServletContext());

String outputString =
  webFigureHtmlGen.getFigureEmbedString(
```

```
    userPlot,
    "UserPlot",
    "session",
    "700",
    "700",
    null);

response.getOutputStream().print(outputString);
```

WebFigure to Bytes

```
byte[] byteArrayFromWebFigure =
                  examples.getByteArrayFromWebFigure();
response.getOutputStream().write(byteArrayFromWebFigure);
```

Raw Image Bytes

```
byte[] rawImageBytes =
           examples.getImageByteArrayFromMWNumericArray();
response.getOutputStream().write(rawImageBytes);
```

Raw Image Bytes with Reorientation

Note: This example allows you to perform similar functionality to the example "Implement a Custom WebFigure", but in a manual implementation. It is one of many ways you can implement this functionality in a stateless manner.

```
int height = Integer.parseInt(request.getParameter("height"));
int width = Integer.parseInt(request.getParameter("width"));
int elevation =
         Integer.parseInt(request.getParameter("elevation"));
int rotation =
          Integer.parseInt(request.getParameter("rotation"));
String imageFormat = request.getParameter("imageFormat");
byte[] rawImageBytes =
  examples.getImageByteArrayFromMWNumericArrayWithOrientation(
    height,
```

```
    width,
    elevation,
    rotation,
    imageFormat);
response.getOutputStream().write(rawImageBytes);
```

3. Inside the staging folder you created at the start of this procedure, create a `WEB-INF` folder.
4. Inside the `WEB-INF` folder, create two additional folders:
   - `classes`
   - `lib`
5. Place all of the class files (including the DAO created in "Creating a DAO for Deployment" on page 4-5) into the `class` folder within the appropriate package folders that exist.
6. Copy the component JAR file into the `lib` folder.
7. Create a `web.xml` file in the `WEB-INF` folder.

   This file provides the Web server with a valid path into your code and defines the entry point into that code. Use this template as an example:

Example of a web.xml File Used in a Java™ Servlet Component

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD
Web Application 2.3//EN"
  "http://www.Oracle.com/servers/dtd/web-app_2_3.dtd">
<web-app>
    <servlet>
        <servlet-name>ExamplesServlet
                              </servlet-name>
        <servlet-class>examples.ExamplesServlet
                             </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ExamplesServlet</servlet-name>
        <url-pattern>/ExamplesServlet</url-pattern>
    </servlet-mapping>
```

```
</web-app>
```

The following URL accesses this servlet with the configuration described above:

```
http://localhost:8080/Examples/ExamplesServlet
```

Note the `Examples` string in the URL, since the JAR is named `Examples.jar`. Using this string sets up the correct server context and is a customizable attribute within the console of many Web servers.

8 Using the `java -jar` command, bundle the folders you created into a WAR (Web archive) and place it in your Web server's component folder.

Note: Some Web servers require you to register the application before it is accessible, usually by referencing the WAR from within the administrator's console.

Hosting a DAO Using a Java™ Web Service

More and more companies are hosting services on the Web, often times with SOAP (Simple Object Access Protocol). This exposes business functions through simple services. Each of these services performs a specific task. Since SOAP is an established standard that is supported by many different languages and third-party applications, it is extremely versatile. You can use a SOAP Web service directly in Microsoft Excel with no prior knowledge of the service's implementation. Multiple language support makes SOAP suitable for use with primitive data types.

Although these primitives can be wrapped in a number of complex object structures, the examples in this section will cover fundamental use cases that should be the same, regardless of data structure and business objects.

In this section, you will learn how to create basic Java objects that handle business logic, while Apache Axis2 performs the mechanics involved with turning the logic a Web service and exposing it. Alternatively, you can start by using WSDL (Web Service Definition Language — the definition of your service) and generate Java from that. Afterward you can customize the Java with your business logic, or change the WSDL manually in a number of other ways to meet your needs.

Setting Up the Root Web Service Class

Since Axis2 supports Java Objects, you will create a shell class to contain all the service methods:

```
package examples;

public class ExamplesWebService
{
  //**************************
  //**Place service methods here
  //**For our examples we will only
  //**be taking in and returning
  //**primitive values
  //**************************
}
```

Interacting with the DAO

Some examples of how to use the DAO with various data types follow:

Numeric

```
public int getInt()
{
  Examples examples = new Examples();
  int integer = examples.getIntFromMWNumericArray();
  examples.dispose();
  return integer;
}
```

String

```
public String getString()
{
  Examples examples = new Examples();
  String string = examples.getStringFromMWCharArray();
  examples.dispose();
  return string;
}
```

Numeric Array

```
public int[] getIntArray()
```

```
{
  Examples examples = new Examples();
  int[] intArray = examples.getIntArrayFromMWNumericArray();
  examples.dispose();
  return intArray;
}
```

Character Array

```
public char[] getCharArray()
{
  Examples examples = new Examples();
  char[] charArray = examples.getCharArrayFromMWCharArray();
  examples.dispose();
  return charArray;
}
```

Byte Array

```
public byte[] getByteArray()
{
  Examples examples = new Examples();
  byte[] byteArray = examples.getByteArrayFromMWNumeric();
  examples.dispose();
  return byteArray;
}
```

Raw Image Bytes

```
Raw Image Bytes
public byte[] getImageByteArray()
{
  Examples examples = new Examples();
  byte[] rawImageBytes =
              examples.getImageByteArrayFromMWNumericArray();
  examples.dispose();
  return rawImageBytes;
}
```

Raw Image Bytes with Reorientation

```
public byte[] reorientAndGetImageByteArray(
  int height,
  int width,
  int elevation,
  int rotation,
  String imageFormat)
{
  Examples examples = new Examples();
  byte[] rawImageBytes =
  examples.getImageByteArrayFromMWNumericArrayWithOrientation(
    height,
    width,
    elevation,
    rotation,
    imageFormat);
  examples.dispose();
  return rawImageBytes;
}
```

Deploying the Web Service

1. Create a staging folder, if one does not exist, and copy the `Examples` DAO class created in "Creating a DAO for Deployment" on page 4-5 and the Web service class created in "Setting Up the Root Web Service Class" on page 4-23into it.
2. Create a `lib` folder and copy your deployed component into it.
3. Create a `meta-inf` folder and, inside it, create a `services.xml` file with these contents:

```
<service>
<parameter name="ServiceClass"
locked="false">examples.ExamplesWebService</parameter>
<operation name="getInt">
<messageReceiver
 class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
</operation>
<operation name="getString">
<messageReceiver
 class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
</operation>
<operation name="getIntArray">
<messageReceiver
```

```
 class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
</operation>
<operation name="getCharArray">
<messageReceiver
 class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
</operation>
<operation name="getByteArray">
<messageReceiver
 class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
</operation>
<operation name="getImageByteArray">
<messageReceiver
 class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
</operation>
</service>
```

The `services.xml` file tells Axis2 which methods to expose, and what mechanism to use to expose them.

4. Copy all of the files into a WAR (Web archive) file and place them in the `axis2` component folder (`axis2/WEB-INF/services`). Use the `java -jar` command but give the output file an `.aar` extension rather than a `.jar` extension.
5. You should now see your service running in the Axis console. From the console, note the URL for the WSDL file. You will use this URL in other applications to communicate with your Web service.

5

Front-End Web Developer Tasks

Working with the Front-End Layer

In this section...

About the Front-End Layer

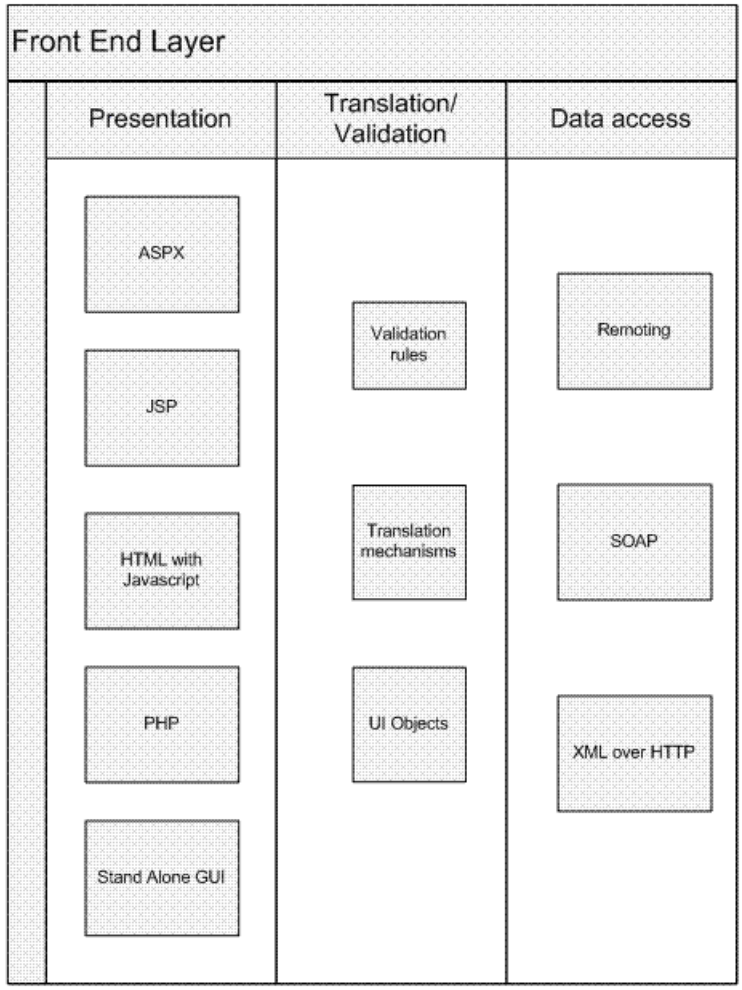
Note: For comprehensive end-to-end implementations of the concepts in this chapter, see Appendix A.

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.



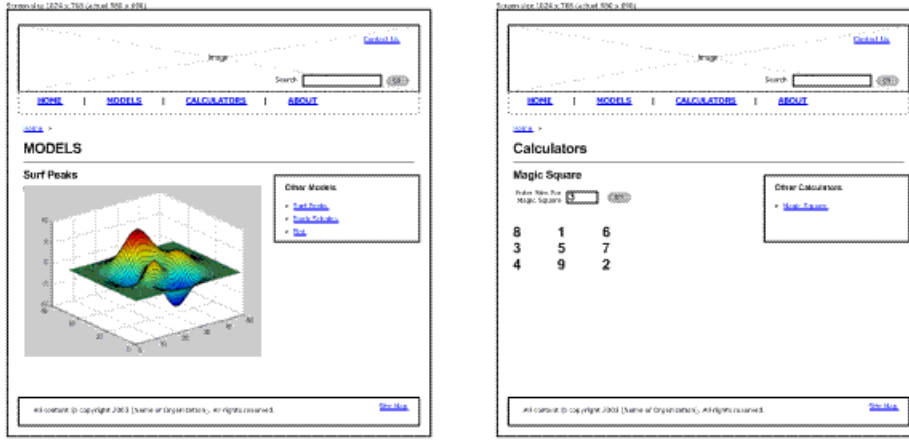
Elements of the Front End Layer

- 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.

Creating a WebFigure on a JSP Page

There are several ways to use WebFigures on a Web page with Java:

- Return a special HTML string from the servlet which embeds the WebFigure into a page.
- Using the custom WebFigure tag library directly from the JSP, have the servlet bind the WebFigure and redirect it to the JSP.
- Generate a string from the middle tier that can be dumped directly onto a Web page at the front end, embedding all of the WebFigure functionality and the correct callbacks directly into the front-end page.

In each case, the WebFigure object is stored in the Web server's session. The Java script that executes the client calls back to the server for updates and, using the cached WebFigure, new updates are sent back to the client.

Using an HTML String

This example is very similar to what you can find in the example “Implement a Custom WebFigure”, but this example also uses the DAO. The middle tier code in “Hosting the DAO with a Servlet” on page 4-16 is repeated here:

```
HttpSession session = request.getSession();

WebFigure userPlot =
      (WebFigure)session.getAttribute("UserPlot");

// if this is the first time doGet has been called for
//  this session,
//  create the plot and WebFigure object
if (userPlot== null)
{
  userPlot = examples.getWebFigureFromMWJavaObjectRef();
  // store the figure in the session context
  session.setAttribute("UserPlot", userPlot);

  // bind the figure's lifetime to the session
  session.setAttribute(
    "UserPlotBinder",
    new MWHttpSessionBinder(userPlot));
```

```
}
```

Using the WebFigure Tag Library

To use the WebFigure object directly from a JSP page, reference the `webfigures` tag library. This creates a WebFigure object with your object's parameters. The `UserPlot` is the name of the Web object that you placed in the cache by the middle tier.

Note: Host the middle tier and the JSP on the same server.

```
<%@ taglib prefix="wf"
       uri=
       "http://www.mathworks.com/builderja/webfigures.tld" %>
< wf:web-figure
  name="UserPlot"
  scope="session"
  root="WebFigures"
  width="100%"
  height="100%"/>
```

When using this approach, the only other code needed on the servlet is a redirect back to the JSP that the above code resides on. In this example, this code is contained in `response.jsp`. The servlet code would look like this:

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

Using Embedded HTML

This option is convenient since all of the “heavy lifting” is done on the server and only a string is sent to the front end. In this example, notice how the servlet is merely referenced and dumps the contents into a Web page frame.

Tip This technique can be used regardless of the transfer protocol or location of the front end or back end. Since a simple string is being sent, you can code the front-end in a number of ways, adapting nicely to a SOAP transfer, for example.

```
<iframe
  src ="http://localhost:8080/Examples/ExamplesServlet?
                          function=webFigureEmbedded"
  width="590"
  height="480">
</iframe>
```

To generate this string, run code similar to this on the servlet:

```
// The argument to the WebFigureHtmlGenerator
// constructor is the URL where the
// WebFigures servlet is mapped (relative to the Web a
// pplication and Servlet context)
WebFigureHtmlGenerator wfHtmlGenerator =
    new WebFigureHtmlGenerator("WebFigures",
                          getServletContext());
String embeddedString;

try
{
//This generates a string that can be sent to the
// response that represents the WebFigure.
embeddedString =
wfHtmlGenerator.getFigureEmbedString(
                         userPlot,
                         "UserPlot",
                         "session",
                         null,
                         null,
                         null));
}
catch(MWException mwe)
{
throw new Exception();
}

response.getOutputStream().println(embeddedString);
```

Working with Static Images

There are several options when dealing with images through a component.

You can simply save the image from the MATLAB code to a drive somewhere using print functionality (the front end references the physical file directly). This is not ideal since the middle tier is behind the firewall (and the front end is in front of it), incurring possible security concerns with where the files reside.

Using Java, you can return a Java image object from MATLAB and use it directly from the JSP or servlet by saving it to disk or converting it to a byte stream.

Using a Static Image In a JSP Page

The simplest option is to return a data stream from your MATLAB function as a byte array — an encoded representation of your image, a common paradigm used when storing and retrieving images from a database. However, it is important to consider that only an `IMG` tag's source can be set, not it's data. The most common solution to this issue is to have the `IMG` tag's source reference a servlet that streams the bytes out through the output stream. Although direct communication between a presentation object and the middle tier usually isn't recommended, in this case it is a good solution. A common implementation is to designate a server that only serves up images, keeping data services and image services separate, as shown here:

```
<img src="http://localhost:
                   8080/Examples/ExamplesServlet?function=
                   imageBytesFromMWNumeric"
                      alt="MATLAB IMAGE" />
```

Interacting with Images Using JavaScript (for .NET or Java)

Although "Creating a WebFigure on a JSP Page" on page 5-6 is a good solution for most component models, sometimes a lightweight solution is needed that you can customize for specific tasks.

JavaScript can be employed to dynamically request new images depending on user input. Since JavaScript is not Java, it does not require that Java Runtime be installed. JavaScript runs in a client's browser and does not require a Java Web server. You can use this lightweight implementation with any of the builders. This example uses the

Raw Image Bytes with Reorientation example in "Hosting a DAO Using a Java™ Web Service" on page 4-23 and "Hosting the DAO with a Servlet" on page 4-16. It waits for the user to instigate a movement with the mouse (a mouse-drag "event") and, when the events occur, calls the server to get a new image of the new orientation. This example, while simple, can be extended to do many other types of image interactions.

```
<iframe
  src ="DynamicFigure.html?url=
    http://localhost:8080/Examples/ExamplesServlet?function=
    imageBytesFromMWNumericWithOrientation"
  width="700"
  height="700">
</iframe>
```

`DynamicFigure.html` is an AJAX application that takes in a parameter (the base function that returns an image) and accepts different orientation values:

```
<%@ page contentType=
                "text/html;charset=UTF-8" language="java" %>
<%@ page isELIgnored ="false"%>
<html>
  <head>
    <title>AJAX Figure Manipulation</title>
    <script type="text/javascript">
      var rotationDegree = 0;
      var elevationDegree = 0;
      var startDragX = 0;
      var startDragY = 0;
      var mouseisdown = false;


      function getParam(name)
      {
       var start=location.search.indexOf("?"+name+"=");
       if (start<0) start=location.search.indexOf("&"+name+"=");
       if (start<0) return '';
       start += name.length+2;
       var end=location.search.indexOf("&",start)-1;
       if (end<0) end=location.search.length;
       var result='';
       for(var i=start;i<=end;i++) {
         var c=location.search.charAt(i);
         result=result+(c=='+'?' ':c);
       }
```

```
        return unescape(result);
       }


       function updateView()
       {
         var urlStr = getParam("url") +"&" +
           "imageFormat=png" + "&" +
           "rotation=" + rotationDegree + "&" +
           "elevation=" + elevationDegree  + "&" +
           "width=" + contentBox.clientWidth + "&" +
           "height=" + contentBox.clientHeight;

         var requestedImage =
                        document.getElementById('currentImage');
         requestedImage.src = urlStr;
         requestedImage.style.visibility = 'visible';
       }

       function stopDragging(updateX,updateY)
       {
        rotationDegree +=
                             Math.round(((startDragX -
                                     updateX)/2)%360);
        elevationDegree +=
                             Math.round(-(startDragY -
                                     updateY)/2);
        updateView();
       }
     </script>
   </head>

  <body onresize='updateView();'>
   <form name=exf1>
    X Drag <input type=text name=x value="0">
    Y Drag <input type=text name=y value="0">
   </form>

   <div style='position:absolute; background:
            url("matlab.gif"); left:0; right:0;
                   width:100%; height:100%;'>
<img style="visibility: hidden;  position:absolute;
            width:100%; height:100%; left:0;
                     top:0" src="matlab.gif"
```

```
id="currentImage">
  </div>

  <div  id='contentBox'
style='position:absolute; background:
                      url("transparent_pixel.gif");
 left:0; top:0; background-color:
                  <%= request.getParameter("color") %>;
 width:100%; height:100%; overflow:hidden;'
   onmousedown="mouseisdown = true; startDragX=event.clientX;
                                    startDragY=event.clientY;"
   onmouseup="mouseisdown =
            false;stopDragging(event.clientX, event.clientY);
            document.exf1.x.value=0;  document.exf1.y.value=0;"
   onmousemove="if(mouseisdown)
            {document.exf1.x.value=event.clientX-startDragX;
             document.exf1.y.value=event.clientY-startDragY;}">
  </div>
  <script type="text/javascript">
   updateView();
  </script>
 </body>
</html>
```

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>");
 }
%>
```

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.

Server Administrator Tasks

Managing a Deployment Server Environment

In this section...

The Server Administrator's Role in Deployment

In many organizations, integration developers do not handle actual production systems. Instead, a group of people that have access to these systems and manage various applications running on shared hardware take responsibility for this task.

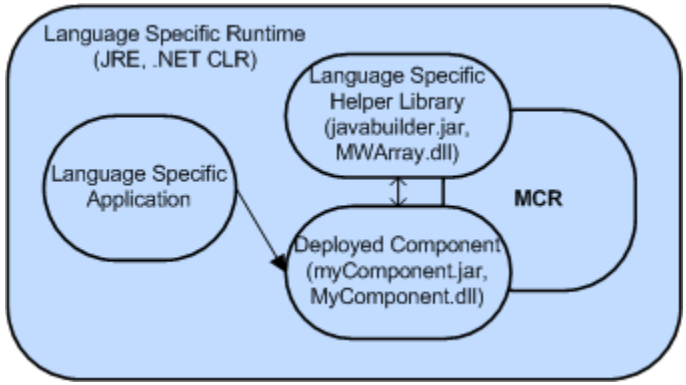
System management requires ensuring that one set of applications does not affect another and also managing version conflicts that arise. Another concern of the server administrator is monitoring system up-time. The following sections discuss the nuances of the deployment products and what the server administrator role needs to do to ensure a successful deployment. It provides different methods of dealing with common issues that can affect stability and performance.

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 and builders. For a deployed application to function against the 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)".

The MATLAB Runtime Installer

The runtime installer ships with MATLAB. You obtain it from the MATLAB programmer who originally compiled the application. The version number of the 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 Builder NE, and MATLAB Builder JA, a Helper Library ships with the runtime. This library contains information for communicating with the 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.

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 runtime's installation directory.

Loading the MATLAB Runtime

Because deployed applications use the MATLAB rumtime at runtime, the runtime must be loaded and running. The 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 (if the runtime resides on a network). To ensure a consistent user experience for Web applications, load the MATLAB rumtime at server start-up time, rather than upon first use, by instantiating a class as part of the server initialization.

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 MCR 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 6-6

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 6-6 to optimize performance.

Session Scaling

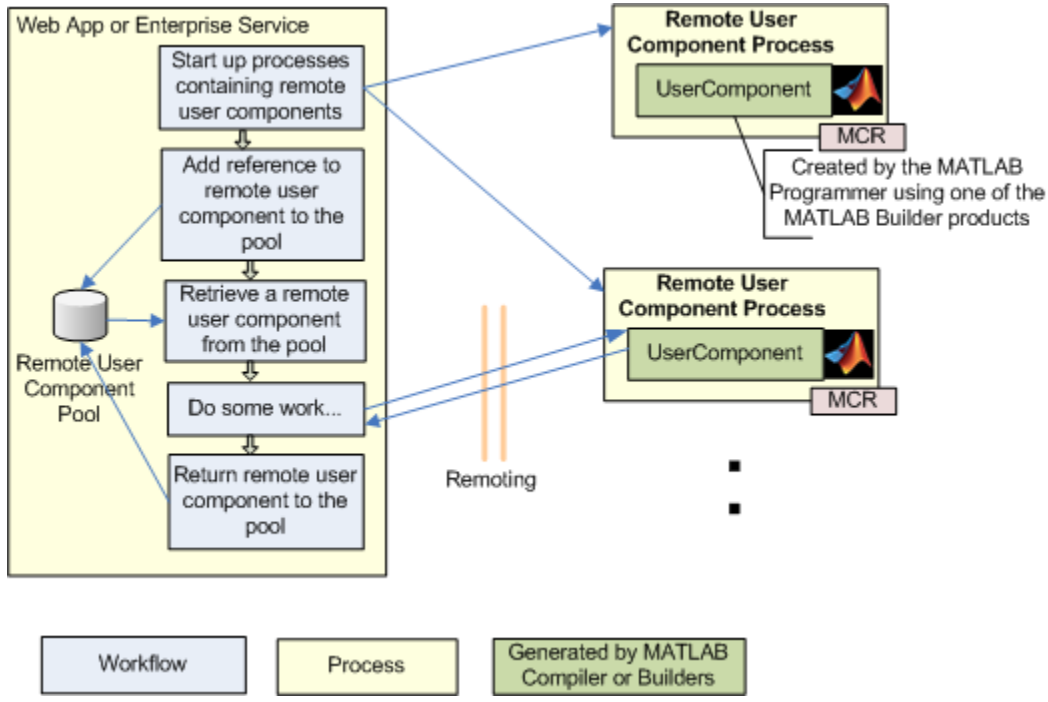
Session scaling involves enabling the maximum number of users while minimizing performance degradation.

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 workaround the situation, enable multiple MATLAB runtimes to service requests as they arrive. Run several MATLAB runtimes in separate processes; one process per 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 “Creating a Remotable .NET Component” 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

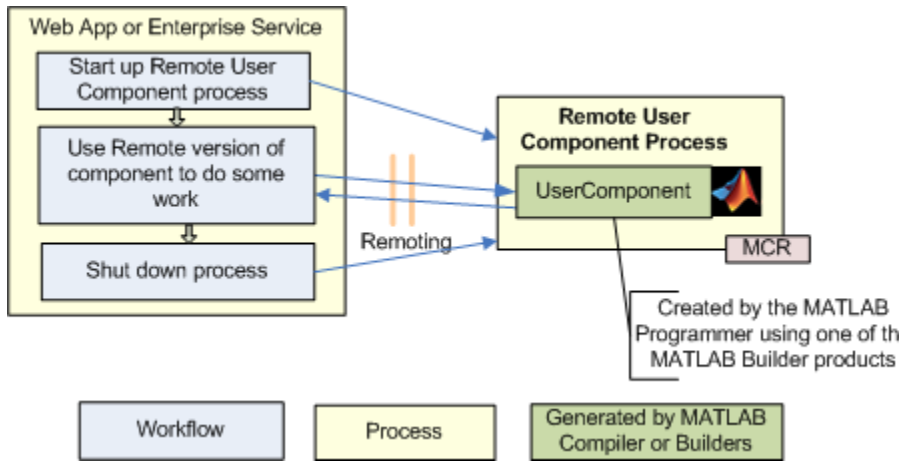
Ensuring Fault Tolerance

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.

Both MATLAB Builder NE and MATLAB Builder JA have 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 robust more application.



Using Remoting to Achieve Fault Tolerance

Hot Deployment

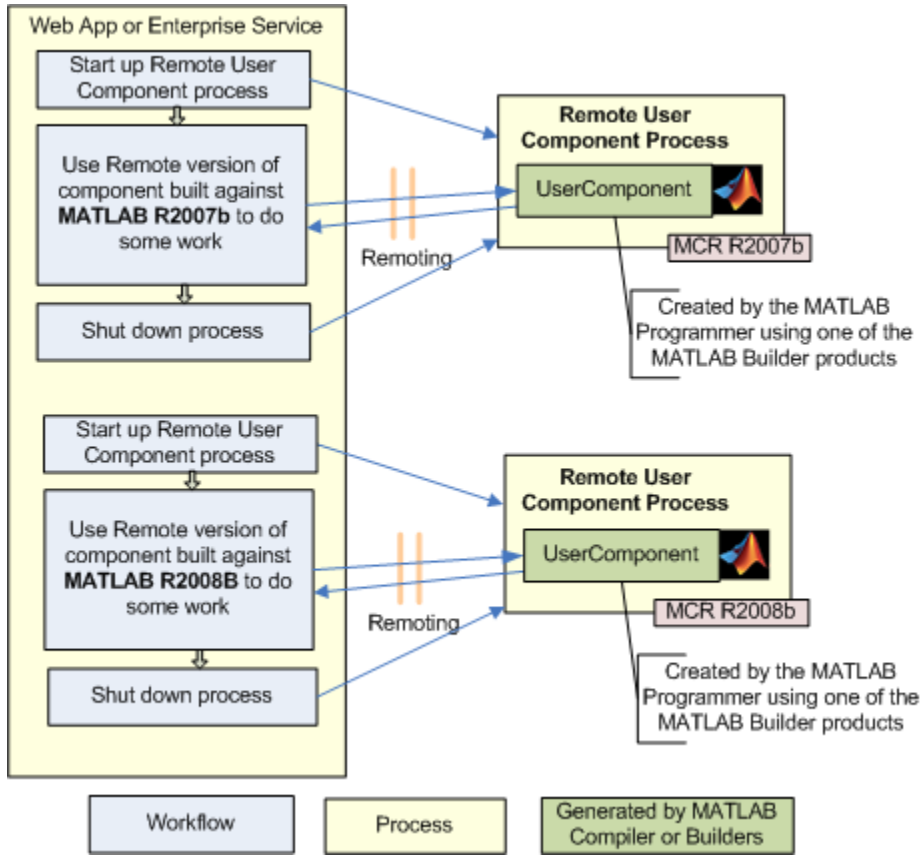
Hot Deployment entails replacing running application source code "on-the-fly." The server automatically migrates end users to this new code without the user experiencing an outage.

Java

Support for Hot Deployment in MATLAB Builder JA is not available at this time.

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.

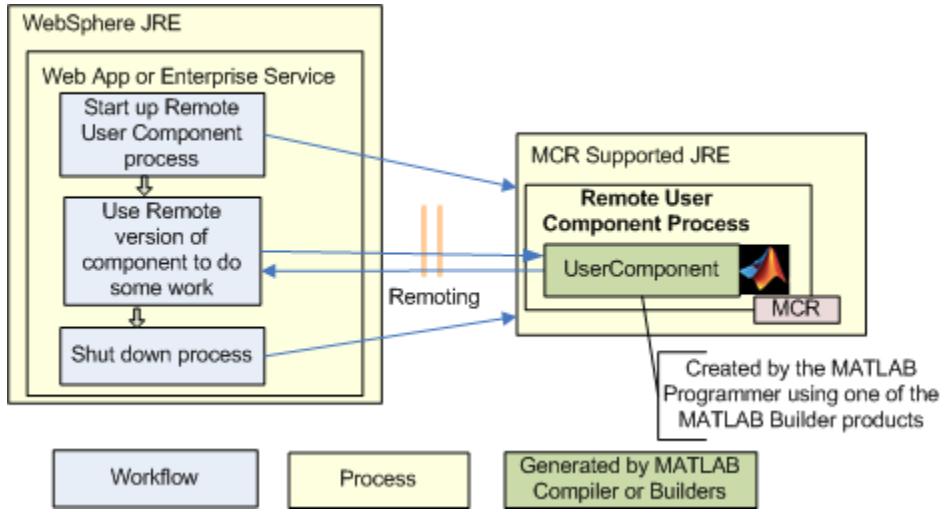


Using Remoting to Workaround Multiple Versions of the MATLAB Runtime

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 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 a 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

7

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.

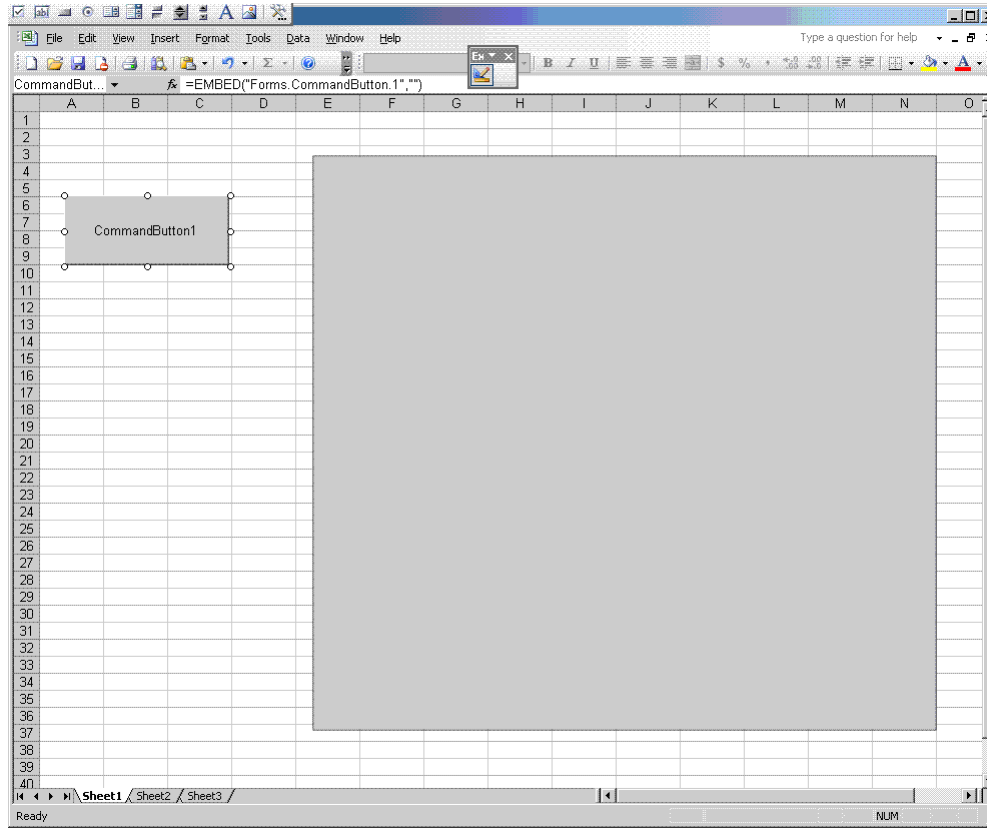
Example Tasks

The example in this section consumes the Web service created in “Deploying the Web Service” on page 4-26. With this type of application, you can use the same Web service to display many different front ends. You can use data stored in Microsoft Excel and pass it to a Web service to generate dynamic data-driven images.

Microsoft Excel Web Service Client Standalone Application

To construct a Microsoft Excel interface to the Web service:

1. Download and install the Microsoft Office Web Service Toolkit from Microsoft, if you haven't already.
2. Start Microsoft Excel.
3. Open a new worksheet.
4. Using the **Control Toolbox**, create an Excel graphics window by dropping and dragging an **Image**.
5. Drag a **Command Button** into the window. You use this button to trigger the Web service call and load the graphic. At this stage, the window looks like this.



6 Double-click the **Command Button** button and the VBA editor starts.

7 Select **Tools > Web Service References**.

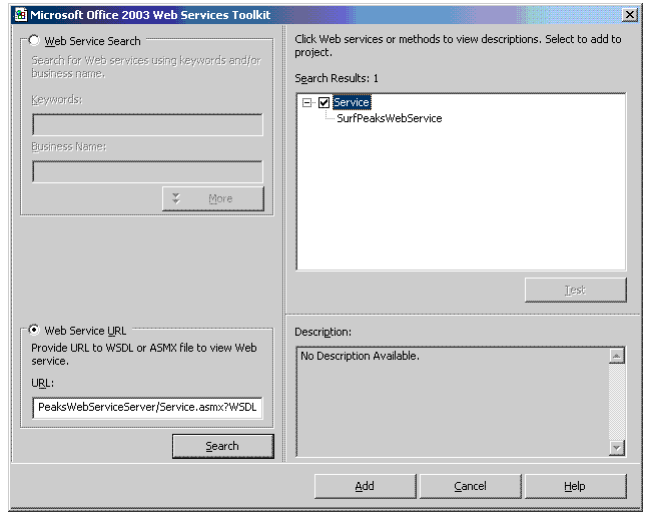
Note: The **Web Service References** option is available only if you install the Microsoft Office Web Service Toolkit.

---

8 In **Web Service URL**, type the WSDL that was referenced in "Using Web Services" on page 5-14:

```
"http://localhost:3465/SurfPeaksWebServiceServer/Service.asmx?WSDL"
```

9 Click **Search** to query the Web service. The result resembles this.



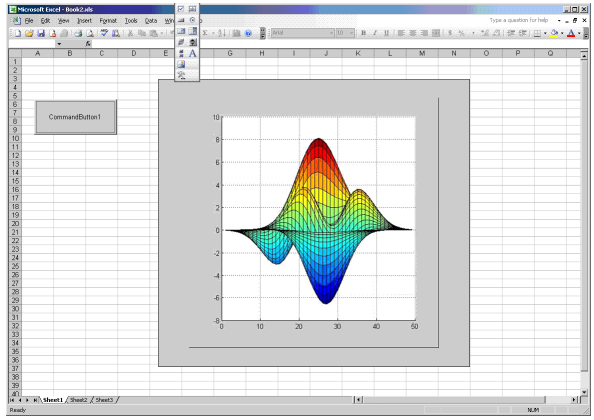
10 Select the appropriate service in the **Search Results** pane and click **Add** to bind it to your project. Notice that a Class Module is created called `clsws_Service`. This module will be used by the button action to retrieve the data.

11 In the worksheet, for the method `CommandButton1_Click()`, add and save the following code:

```
Sheet1.Image1.Picture = Nothing

Dim value() As Byte
Set module = New clsws_Service
value = module.wsm_SurfPeaksWebService

'Saves byte() data from web service to a file
Dim intFileNumber As Integer
intFileNumber = FreeFile
Open "c:\temp1.png" For Binary As #intFileNumber
Put intFileNumber, , value
Close #intFileNumber

'Loads the saved picture into the image
Sheet1.Image1.Picture = LoadPicture("c:\temp1.png")
```

12 Click the command to display the following in the graphics pane of your worksheet.

Tip You may need to close Microsoft Excel and reopen it to see the graphic.



8

End-to-End Developer Tasks

Role of the End-To-End Developer

Each chapter in this guide is focused on tasks that are performed in order to deploy applications from the perspectives of various types of users. This is done by providing snippets of code that a person in a particular role can use to solve a particular problem within the context of some larger application. While this approach makes sense for most users, sometimes a single user is fulfilling *all* roles, and often this person is relatively new to some of the roles. For example, sometimes an expert MATLAB programmer is asked to put something they've worked on up to the Web for others to consume. They may never have used Java or .NET before.

This chapter is aimed specifically at users playing the role of the "one-stop shop" and will go through in relative detail all of the steps needed to build an application from the ground up and get it running successfully.

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.

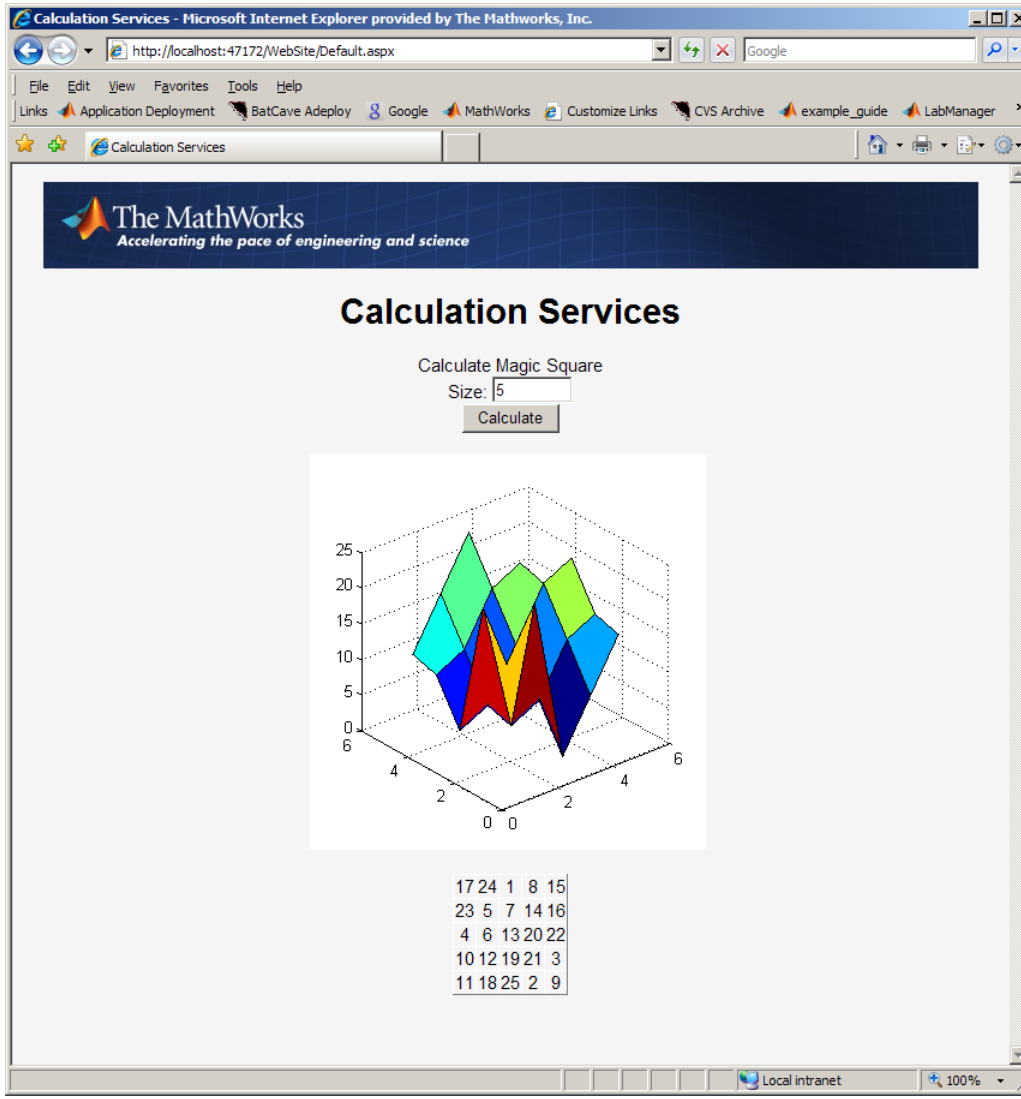
The `getMagicWebFigure.m` MATLAB function, which runs the Magic Square calculator, is as follows. It is based on the popular Magic Square `getMagic` function:
getmagic.m:

```
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

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 8-3 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 MATLAB Builder JA JAR file that is generated by compiling MATLAB code using MATLAB, MATLAB Compiler, and MATLAB Builder JA.
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 8-5.

Building Your Example Component

1. Download the application code for this example from MATLAB Centrals File Exchange at `http://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 8-5.

```
<%@ 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 ..
```

A

Sources for More Information

Other Examples

Use these links for more information on other Web examples of possible interest:

MATLAB Builder JA

Other examples using MATLAB Builder JA include:

Black-Scholes

MATLAB Central Black-Scholes Web Application for Java

WebFigures

MATLAB Builder JA WebFigures Varargs Application