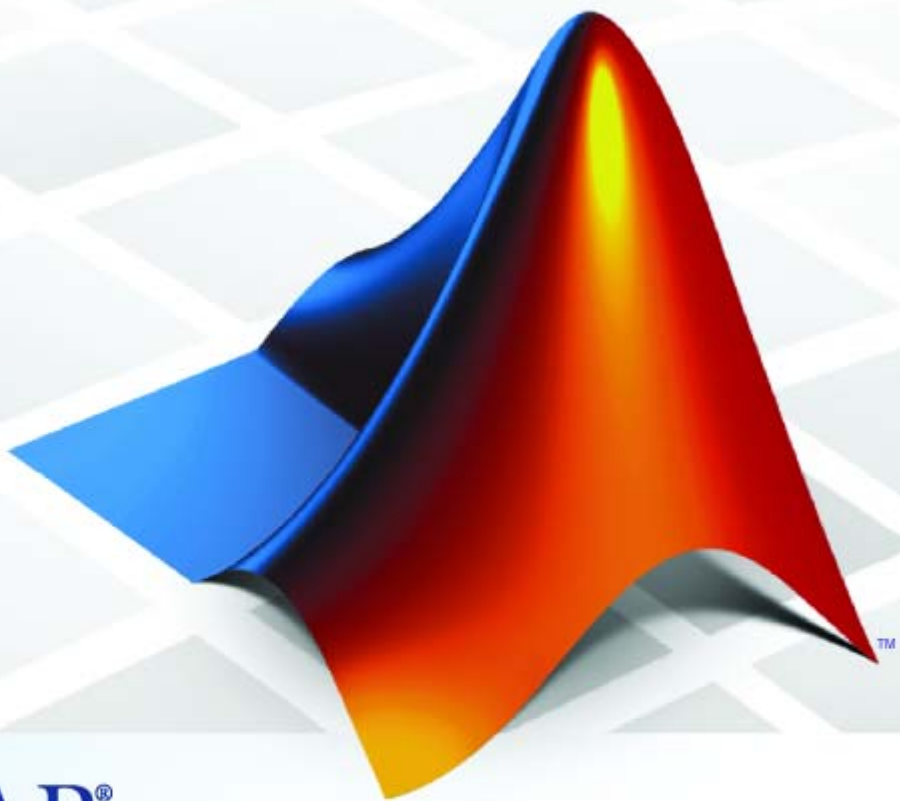


MATLAB® Application Deployment Web Example Guide



MATLAB®

How to Contact The MathWorks



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



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



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

MATLAB® Application Deployment Web Example Guide

© COPYRIGHT 2008–2010 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

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

Patents

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

Revision History

March 2008	Online only	New for MATLAB® Compiler™ Version 4.8 (Release R2008a)
October 2008	Online only	Revised for MATLAB® Compiler™ Version 4.9 (Release R2008b)
March 2009	Online only	Revised for MATLAB® Compiler™ Version 4.10 (Release R2009a)
September 2009	Online only	Revised for MATLAB® Compiler™ Version 4.11 (Release R2009b)
March 2010	Online only	Revised for MATLAB® Compiler™ Version 4.13 (Release R2010a)

How to Use This Guide

1

About This Guide	1-2
Who Should Use This Guide?	1-3
Commonly Used Software and Preliminary Setup	
Information	1-4
MATLAB Programmer	1-4
Integration Experts (Business Service Developer and Front-End Developer)	1-4
Internal Analyst	1-5

Web 101: An Introduction to Web Concepts and Terminology

2

Why Learn About Web Concepts and Terminology? ...	2-2
Basics of Web Processing	2-3
HTML Documents and HTTP	2-3
Integrating MATLAB Code on the Web Using Java Servlets	2-4
Optimizing Performance of Servlets That Call Your MATLAB Web Code	2-6
Maintaining Data Integrity Through Proper Scoping	2-9
Lifecycle of a Deployed MATLAB Web Application	2-16
Introduction	2-16
MATLAB Web Application Environment	2-18
MATLAB Programmer	2-21
Business Service Developer	2-21

Front-End Developer	2-21
Server Administrator	2-22
Internal Analyst	2-22
End-To-End Developer	2-22

MATLAB Programmer Tasks

3

Programming in MATLAB	3-2
Returning Data Types	3-3
Deploying MATLAB Code with the Builders	3-6

Business Service Developer Tasks

4

Working with the Business Service Layer	4-2
About the Business Service Layer	4-2
About the Examples	4-4
Creating a DAO for Deployment	4-6
Initializing a Component	4-6
Interacting with a Component	4-7
Hosting a DAO on a Web Server	4-24
Hosting the DAO with a Servlet	4-24
Hosting a DAO Using a Java™ Web Service	4-31
Hosting a .NET DAO with ASPX	4-36
Hosting a DAO Using a .NET Web Service	4-40

Front-End Developer Tasks

5

Working with the Front-End Layer	5-2
About the Front-End Layer	5-2
About the Examples	5-4
Creating a WebFigure on a JSP Page	5-6
Creating a WebFigure on an ASPX Page	5-10
Working with Static Images	5-13
Displaying Complex Data Types Including Arrays and Matrices	5-18
Using Web Services	5-26

Server Administrator Tasks

6

Managing a Deployment Server Environment	6-2
The Server Administrator's Role in Deployment	6-2
An Overview of Deployed Applications	6-2
Installing the MATLAB Compiler Runtime (MCR)	6-3
Loading the MATLAB Compiler Runtime (MCR)	6-6
Scaling Your Server Environment	6-8
Ensuring Fault Tolerance	6-9
Hot Deployment	6-11
.NET	6-11
Java	6-11
Working with Multiple Versions of the MATLAB Compiler Runtime (MCR)	6-12

Unsupported Versions of the JVM	6-13
--	-------------

Internal Analyst Tasks

7

Working with Content	7-2
Example Tasks	7-3

End-to-End Developer Tasks

8

Role of the End-To-End Developer	8-2
Example: The Magic Square Calculator On the Web ..	8-3
Creating an End-to-End Web Application	8-5
Creating a Java Web Application, End-to-End	8-5
Creating a .NET Web Application, End-to-End	8-13

Sources for More Information

A

Other Examples	A-2
MATLAB® Builder JA	A-2
MATLAB® Builder NE	A-2

Index

How to Use This Guide

- “About This Guide” on page 1-2
- “Who Should Use This Guide?” on page 1-3
- “Commonly Used Software and Preliminary Setup Information” on page 1-4

About This 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.

When deploying a MATLAB application in a Web-based environment, the number of possible architectural configurations, programming languages, operating systems, Web servers, and frameworks can be intimidating, especially if you are new to Web deployment.

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.

Use this 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 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.

Commonly Used Software and Preliminary Setup Information

In this section...
“MATLAB Programmer” on page 1-4
“Integration Experts (Business Service Developer and Front-End Developer)” on page 1-4
“Internal Analyst” on page 1-5

MATLAB Programmer

- 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 (Business Service Developer and Front-End Developer)

- MCR (MATLAB® Compiler™ Runtime)
- Microsoft® Visual Studio™ 2005 or other supported IDE
- 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

Internal Analyst

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

		Business	Integration Experts		End User		End-to-end User
Roles		 MATLAB Programmer	 Business Service Developer	 Front End Developer	 External User	 Internal Analyst	 General Developer
	Software			MCR, Java web server (e.g., Tomcat), Soap server (e.g., Axis)	Java web server, home grown apps (e.g., Excel), third party apps (e.g., Siebel)	Web browser (Firefox, IE, Safari)	Web browser (Firefox, IE, Safari)
				MCR, IIS, Soap server (IIS)	IIS, home grown apps (e.g., Excel), third party apps (e.g., Siebel)		Excel

Web 101: An Introduction to Web Concepts and Terminology

- “Why Learn About Web Concepts and Terminology?” on page 2-2
- “Basics of Web Processing” on page 2-3
- “Lifecycle of a Deployed MATLAB Web Application” on page 2-16

Why Learn About Web Concepts and Terminology?

Interest in Web concepts increases proportionally in a business organization as the technology becomes more vital to the success of that business.

To ensure the success of the organization, your team members must understand basic concepts and terminology relating to Web information processing. In organizations where resources are limited, Web infrastructure and deployment tasks are often assumed by a limited number of staff (the “one-stop-shops” mentioned in Chapter 8, “End-to-End Developer Tasks”). In such environments, understanding the basics becomes even more vital to success.

This introduction includes:

- “Basics of Web Processing” on page 2-3 — What happens after a URL is entered or a GUI button on a Web page is clicked? This topic details the basics of Java servlet-based Web processing. You’ll also read about the hardware and software used in Web architecture, and the terminology that surrounds Web-based applications.
- The Lifecycle of a Deployed MATLAB® Web Application — This topic introduces you to the various tiers of infrastructure that comprise the MATLAB Web deployment environment and the most prominent deployment roles played

Basics of Web Processing

In this section...

“HTML Documents and HTTP” on page 2-3

“Integrating MATLAB Code on the Web Using Java Servlets” on page 2-4

“Optimizing Performance of Servlets That Call Your MATLAB Web Code” on page 2-6

“Maintaining Data Integrity Through Proper Scoping” on page 2-9

HTML Documents and HTTP

Web processing takes place on some combination of a client, running a Web browser (such as Microsoft®Internet Explorer® or Mozilla® Firefox®) and a Web server. HTTP (HyperText Transfer Protocol) is a simple, synchronous protocol—a client connecting to a server, sending a single command, then waiting for a response from the server.

In the most basic scenario, the URL specifies the exact document to be read. A browser using HTTP retrieves the document for display. Most Web applications, however, deal extensively with dynamic, interactive content.

Typically, Web processing is made up of the following workflow:

- 1** The user enters a URL request, either manually or via a search engine, in the browser.
- 2** The browser parses the URL into two parts: one specifying the server and one specifying a document on the server to be retrieved.

For example, the URL `http://www.mathworks.com/product-list.html` indicates the `product-list` HTML document on The MathWorks™ server.

- 3** A browser using HTTP retrieves the document from the server. After the data is downloaded into the browser, it is parsed and rendered. The document may refer to other documents (via hyperlinked URLs) which are similarly processed until the full Web page is rendered.

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 servlet class. For example, class `MyServlet` extends `HttpServlet` (part of the Java Servlets API). In the `doGet` method of `MyServlet`, you make calls to a deployed component.

In the “Deploying a Component with the Magic Square Example”, for example, you instantiate the `MagicDemoJavaApp` component inside `doGet` (taking care to call the `dispose` method to release native resources).

- 2** Write code to respond to the client request. At minimum, you must at least set the content type (HTML, for example).
- 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 `MWArrays` to strings or another type appropriate to your application.

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.PrintWriter;

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 PrintWriter page = new PrintWriter(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, `MagicDemo` is instantiated along with its `dispose` method.

This can quickly impact performance, as it takes time to create an MCR 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`'s `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 `MagicDemoJavaApp` to the scope of the servlet. This renders the lifetime of the deployed component and its MCR 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 2-6, `MyServlet` takes a request and generates an HTML document calling `MagicDemoJavaApp`. This call generates a formatted response. However, you may want a different formatted response or you may want to call `MagicDemoJavaApp` 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 GETs 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 `MagicDemoJavaApp`.

Because each of the servlets gets their `init` method called separately, each of these methods initializes *a separate MCR 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 2-10.

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 `MagicDemoInstance`:

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

The servlet container also 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 requests, 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`, `removeAttribute`, and so on. They bind variables that last for the relationship between a specific client with browser and server.

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

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

MyHttpSessionListener.java.

```
import magicsquare.magic;

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

import com.mathworks.toolbox.javabuilder.MWException;

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

    public void sessionCreated(final HttpSessionEvent event) {
        final HttpSession session = event.getSession();
        try {
            final magic m = new magic();
            session.setAttribute(SHARED_MAGIC_INSTANCE, m);
        } catch (MWException e) {
            session.getServletContext().log("Error creating magicsquare.magic", e);
        }
    }

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

```
}
```

MyServer.java.

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

(magic)session.getAttribute(MyHttpSessionListener.SHARED_MAGIC_INSTANCE);
```

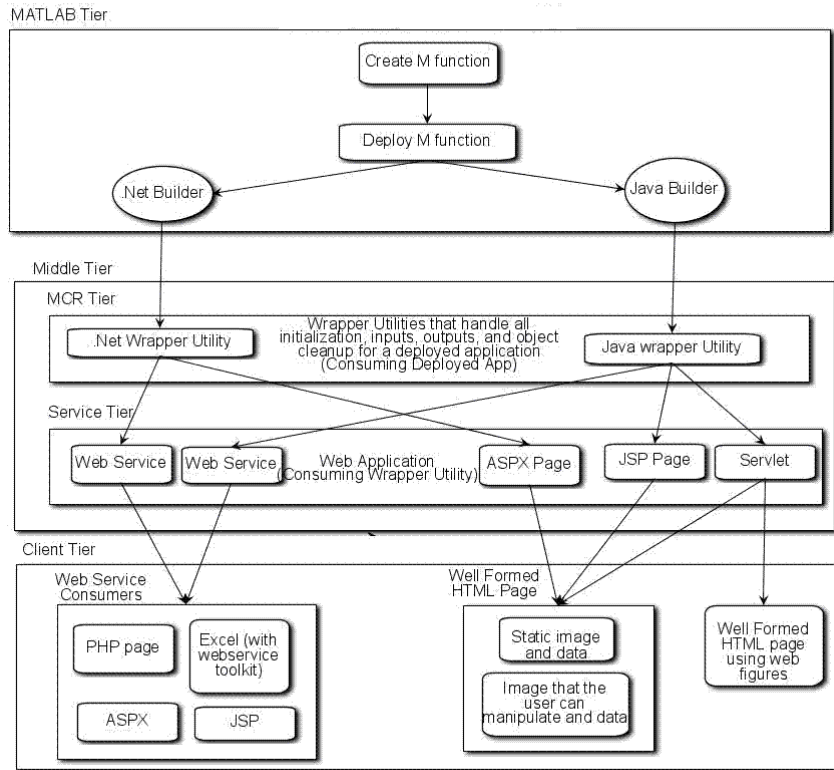
Lifecycle of a Deployed MATLAB Web Application

In this section...
“Introduction” on page 2-16
“MATLAB Web Application Environment” on page 2-18
“MATLAB Programmer” on page 2-21
“Business Service Developer” on page 2-21
“Front-End Developer” on page 2-21
“Server Administrator” on page 2-22
“Internal Analyst” on page 2-22
“End-To-End Developer” on page 2-22

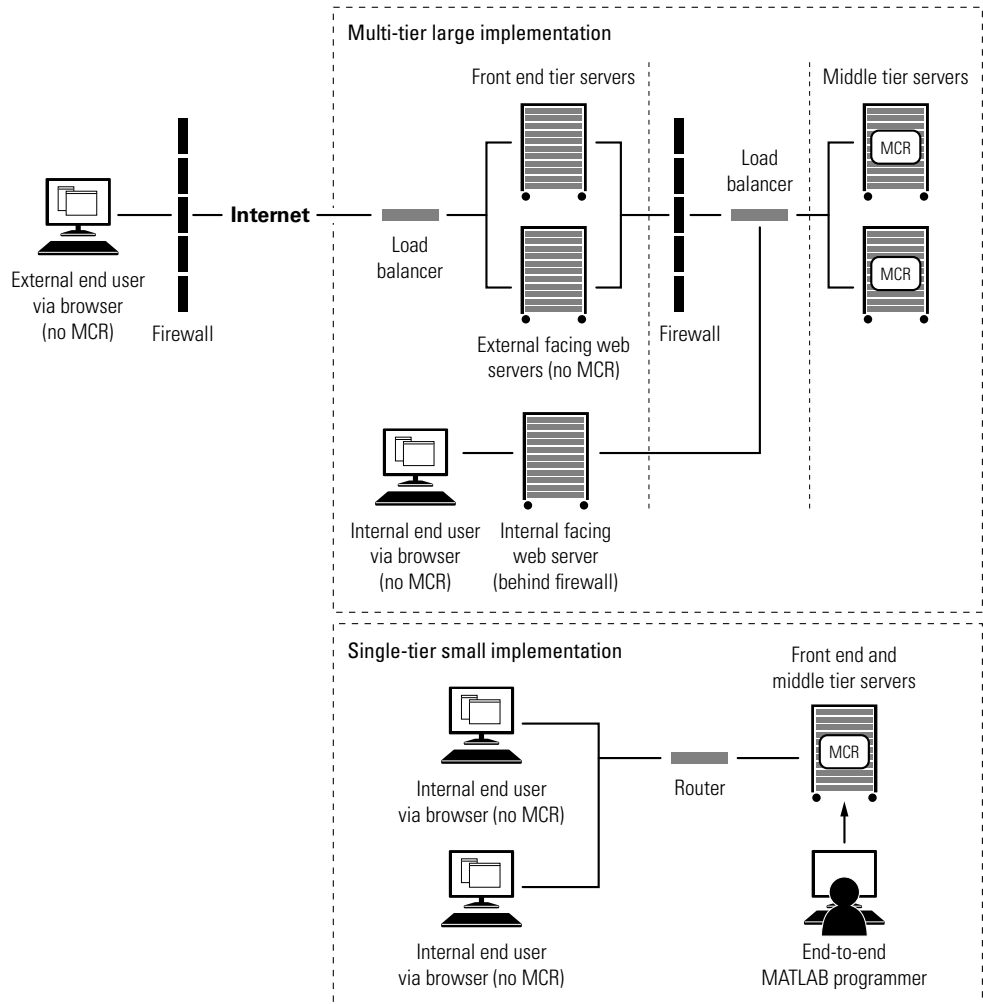
Introduction

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.



Deployed Application Tier Interactions









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 Compiler Runtime (MCR), which is a set of libraries that runs encrypted MATLAB code.

In a Web application, the builder products allow integration of the MCR 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.

	Role	Knowledge Base	Responsibilities
Business	 MATLAB Programmer	MATLAB expert No IT experience No access to IT systems	Develops model Uses the builder tools to create a component that is given to the business service developer
Integration Experts	 Business Service Developer	No MATLAB experience Creates services out of components received from the MATLAB programmer	Creates reusable service that is scalable
	 Front End Developer	Service consumer responsible for presentation and usability No MATLAB experience	Creates front end applications
End User	 External User	External user over the internet through firewalls No MATLAB or IT experience	Customer of the company User of the site
	 Internal Analyst	Internal user over the network Little to no MATLAB or IT experience	Employer of the company User of tools and sites
End-to-End Developer	 General Developer	MATLAB expert Creates services out of components received from the MATLAB programmer Service consumer responsible for presentation and usability	Develops model Uses the builder tools to create a component that is given to the business service developer Creates reusable service that is scalable Creates front end applications

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 business service developer.

Business Service 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 MCR, 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 MCR 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 Developer

Front-end 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 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. External 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.

Internal Analyst

Internal analysts 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 business service developer and front-end 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.

MATLAB Programmer Tasks

- “Programming in MATLAB” on page 3-2
- “Deploying MATLAB Code with the Builders” on page 3-6

Programming in MATLAB

 <p>MATLAB Programmer</p>	<p>MATLAB expert</p> <p>No IT experience</p> <p>No access to IT systems</p>	<p>Develops model</p> <p>Uses the builder tools to create a component that is given to the business service developer</p>
---	---	---

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 the documentation for the product you are using (MATLAB Compiler, MATLAB Builder JA, or MATLAB Builder NE)

Creating a Deployable MATLAB Function

Virtually any calculation that you can create in MATLAB can be deployed, providing it is contained in a function. For example:

```
>> 1 + 1
```

cannot be deployed.

However,

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

can be deployed, since 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 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 “Deploying a Java Component Over the Web” in the MATLAB Builder JA User’s Guide or “Deploying a MATLAB Figure Over the Web Using WebFigures” in the MATLAB Builder NE User’s Guide), the respective builder marshalls 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 Deployment Tool (`deploytool`), see “Getting Started” in the MATLAB Builder JA documentation and “Getting Started” in the MATLAB Builder NE documentation.


It is also possible to use the MATLAB® Compiler™ `mcc` command to build components. See the `mcc` command reference page in any of the builder products for more information.

Business Service Developer Tasks

- “Working with the Business Service Layer” on page 4-2
- “Creating a DAO for Deployment” on page 4-6
- “Hosting a DAO on a Web Server” on page 4-24

Working with the Business Service Layer

In this section...
“About the Business Service Layer” on page 4-2
“About the Examples” on page 4-4

 Business Service Developer	No MATLAB experience Creates services out of components received from the MATLAB programmer	Creates reusable service that is scalable
--	--	---

About the Business Service Layer

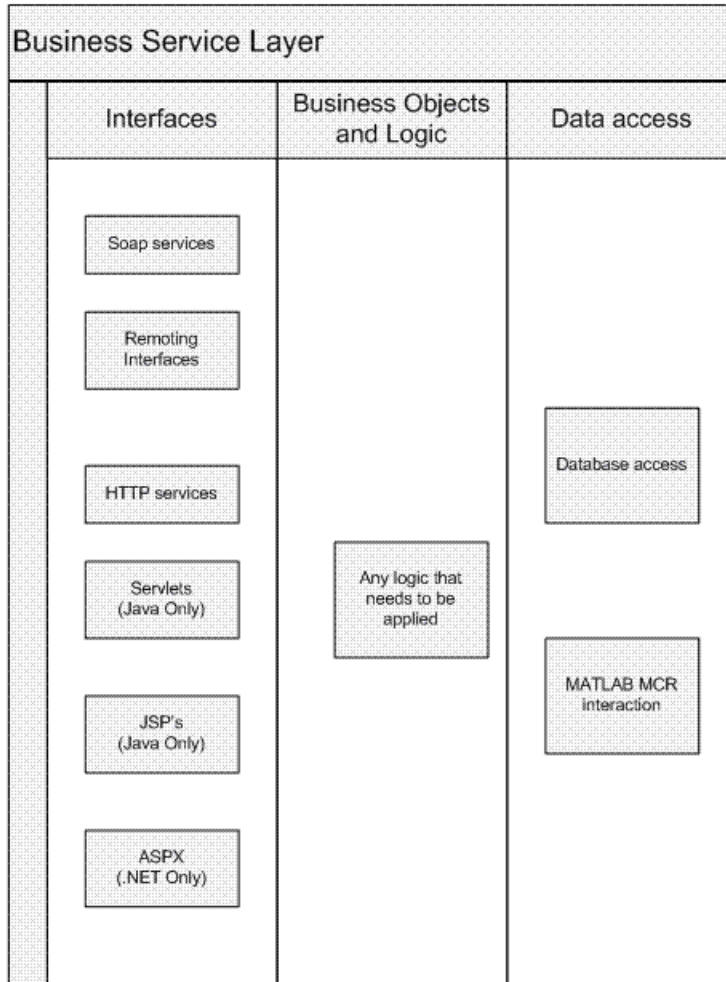
Note For comprehensive end-to-end implementations of the concepts in this chapter, see Appendix A, “Sources for More Information”.

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 Compiler Runtime (MCR).



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();
}
```


.NET

```

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

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

```

Interacting with a Component

You interact with a component by passing inputs to a deployed application or producing MATLAB output from a deployed application. All of these examples fit where the comment block resides in “Initializing a Component” on page 4-6 and the same component class is used. The Java and .NET Builder infrastructure handles data marshalling 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);

```

.NET

```

int n = 3;

```

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

Converting Array Data to a MATLAB Data Type

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

Java

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

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

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

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

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

.NET

```
String[] array =
{
```

```

        "Jordan Robert",
        "Mary Smith",
        "Stacy Flora",
        "Harry Alpert"
    };

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

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

```

Converting a List to a MATLAB Data Type

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

Java

```

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

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

//Create a MWCellArray to store the data
MWCellArray cellArray2 =
    new MWCellArray(

```

```
        numberOfListColumns,  
        numberOfListElements);  
  
//Iterate through the list adding the elements  
//    to the cell array.  
  
Iterator friendsListItr = friendsList.iterator();  
for(int i = 1;friendsListItr.hasNext(); i++)  
{  
    String currentFriend = (String)friendsListItr.next();  
    cellArray2.set(i, currentFriend);  
}
```

.NET

```
List<String> list = new List<String>();  
list.Add("Jordan Robert");  
list.Add("Mary Smith");  
list.Add("Stacy Flora");  
list.Add("Harry Alpert");  
  
int numberOfArrayElements = list.Count;  
int numberOfArrayColumns = 1;  
  
MWCellArray cellArray =  
    new MWCellArray(  
        numberOfArrayColumns,  
        numberOfArrayElements);  
int i = 1;  
foreach (String currentElement in list)  
{  
    cellArray[i] = currentElement;  
    i++;  
}
```

Converting Name Value Pairs to a MATLAB Data Type

Java (Maps)

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

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

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

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

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

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

.NET (Dictionaries)

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

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

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

Getting MATLAB Numerics from a Deployed Application

This code resides in the try block for an initialized component (see “Initializing a Component” on page 4-6).

Java

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

```
}  
finally  
{  
    MWArray.disposeArray(numericArray);  
}
```

.NET

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

Getting MATLAB Strings from a Deployed Application

Java

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

.NET

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

Getting MATLAB Numeric Arrays from a Component

Java

```
Object[] numericArrayOutput = null;
```

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

.NET

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

Getting Character Arrays from a Component

Java

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



```

    MWArray.disposeArray(mwCharArray);
}

```

.NET

```

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

```

Getting Byte Arrays from a Component

Java

```

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

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

```

.NET

```

MWNumericArray result =

```

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

Getting Cell Arrays from a Component

Java

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

```
Object[] cellArrayOutput = null;  
MWCellArray cellArray = null;  
try  
{  
    cellArrayOutput = deployment.getCellArray();  
    cellArray = (MWCellArray)cellArrayOutput[0];  
  
    List listOfCells = new LinkedList();  
    Object[] arrayOfCells =  
        new Object[cellArray.numberOfElements()];  
  
    for(int i = 0; i < cellArray.numberOfElements(); i++)  
    {  
        Object currentCell = cellArray.getCell(i + 1);  
  
        listOfCells.add(currentCell);  
        arrayOfCells[i] currentCell;  
    }  
}  
finally  
{  
    MWArray.disposeArray(cellArray);  
}
```

.NET

```
MWCellArray result = (MWCellArray)deployment.getCellArray();  
  
List<Object> outputList = new List<Object>();
```

```

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

```

Getting Structures from a Component

Java

```

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

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

    Map mapOfStruct = new HashMap();

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

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

```

.NET

```

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

```

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

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

Java

For more information about WebFigures, see “Deploying a Java Component Over the Web” in the *MATLAB Builder JA User’s Guide*.

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

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

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

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

.NET

For more information about WebFigures, see “Deploying a MATLAB Figure Over the Web Using WebFigures” in the *MATLAB Builder NE User’s Guide*. The first two lines of code will not do anything if executed directly. It needs a WebFigureControl on a front end page that is called WebFigureControl1. If you are not using a local WebFigureControl and want to simply return the WebFigure in the current response object, use this code, for example:

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

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

String localEmbedString =
    WebFigureServiceUtility.GetHTMLEmbedString(
        "SessionStateWebFigure",
        WebFigureAttachType.session,
        300,
        300);
Response.Write(localEmbedString);
```

Getting Encoded Image Bytes from an Image in a Component

Java

```
public byte[] getByteArrayFromDeployedComponent()
```

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

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

.NET

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

    MWNumericArray result =
        (MWNumericArray)deployment.getImageDataOrientation(
            height,
            width,
            elevation,
            rotation,
```

```

        imageFormat);
    return (byte[])result.ToVector(MWArrayComponent.Real);
}

```

Getting a Buffered Image in a Component

Java

```

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

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

public BufferedImage getBufferedImageFromDeployedComponent()
{
    try
    {
        byte[] imageByteArray = getByteArrayFromDeployedComponent();
        return ImageIO.read(new ByteArrayInputStream(imageByteArray));
    }
}

```

```
    }  
    catch(IOException io_ex)  
    {  
        io_ex.printStackTrace();  
    }  
}
```

.NET

```
public byte[] getByteArrayFromDeployedComponent()  
{  
    MWArray width = 500;  
    MWArray height = 500;  
    MWArray rotation = 30;  
    MWArray elevation = 30;  
    MWArray imageFormat = "png";  
  
    MWNumericArray result =  
        (MWNumericArray)deployment.getImageDataOrientation(  
            height,  
            width,  
            elevation,  
            rotation,  
            imageFormat);  
    return (byte[])result.ToVector(MWArrayComponent.Real);  
}  
  
public Image getImageFromDeployedComponent()  
{  
    byte[] byteArray = getByteArrayFromDeployedComponent();  
    MemoryStream ms = new MemoryStream(myByteArray, 0,  
    myByteArray.Length);  
    ms.Write(myByteArray, 0, myByteArray.Length);  
    return Image.FromStream(ms, true);  
}
```


Getting Image Data from a WebFigure

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

.NET

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

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

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

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

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

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 4-6. 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 MCR 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 than 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 “Deploying a Java Component Over the Web”, but this example 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 what WebFigures (see “Deploying a Java Component Over the Web” in the MATLAB Builder JADocumentation) performs, 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.getImageByteArrayFromMMNumericArrayWithOrientation(
        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 4-6) 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://java.sun.com/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 POJOs (Plain Old 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 4-6 and the Web service class created in “Setting Up the Root Web Service Class” on page 4-32 into 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.

Hosting a .NET DAO with ASPX

Initializing the DAO

Before a DAO can be used, it must be initialized. The basic template to initialize a .NET DAO looks like this:

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Examples.Examples examples = new Examples.Examples();

        //*****
        /**This is where the examples would be pasted in
        //*****
        //for Examples:
        int integer = examples.getIntFromMWNumericArray();
        Response.Write("int: " + integer);

        examples.dispose();
    }
}
```

Interacting with the DAO

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

Numeric

```
int integer = examples.getIntFromMWNumericArray();
Response.Write("int: " + integer);
```

String

```
String stringResult = examples.getStringFromMWCharArray();
Response.Write("String: " + stringResult);
```

Double Array

```
double[] doubleArray =
    examples.getDoubleArrayFromMWNumericArray();
Response.Write("Double Array: ");
for (int i = 0; i < doubleArray.Length; i++)
{
    Response.Write("Array index(" + i + "): " + doubleArray[i]);
}
```

Character Array

```
char[] charArray = examples.getCharArrayFromMWCharArray();
Response.Write("Char Array: ");
for (int i = 0; i < charArray.Length; i++)
{
    Response.Write("Array index("+ i + "): " + charArray[i]);
}
```

Cell Array to Array

```
Object[] array = examples.getArrayFromCellArray();
for (int i = 0; i < array.Length; i++)
{
    Response.Write("Array index("+ i + "): " + array[i]);
}
```

```
}
```

Cell Array to List

```
List<Object> list = examples.getListFromCellArray();  
foreach (Object currentObj in list)  
{  
    Response.Write("List Item: " + currentObj);  
}
```

Structure

```
Dictionary<Object, Object> dictionary =  
    examples.getDictionaryFromStruct();  
Response.Write("Structure Array: ");  
foreach (Object currentKey in dictionary.Keys)  
{  
    Response.Write("Key: " + currentKey + "    Value: " +  
        dictionary[currentKey]);  
}
```

Byte Array

```
byte[] byteArray = examples.getByteArrayFromMWNumericArray();  
Response.Write("Byte Array: ");  
for (int i = 0; i < byteArray.Length; i++)  
{  
    Response.Write(byteArray[i]);  
}  
Response.BinaryWrite(byteArray);
```

Raw Image Bytes

```
byte[] rawImageBytes =  
    examples.getImageByteArrayFromMWNumericArray();  
Response.BinaryWrite(rawImageBytes);
```


Raw Image Bytes with Reorientation

Note This example allows you to perform similar functionality to what WebFigures performs, but in a manual implementation. It is one of many ways you can implement this functionality in a stateless manner.

```
int height = Convert.ToInt32(Request.Params.Get("height"));
int width = Convert.ToInt32(Request.Params.Get("width"));
int elevation =
    Convert.ToInt32(Request.Params.Get("elevation"));
int rotation = Convert.ToInt32(Request.Params.Get("rotation"));
String imageFormat = Request.Params.Get("imageFormat");
byte[] rawImageBytes =
    examples.getImageByteArrayFromMWNumericArrayWithOrientation(
        height,
        width,
        elevation,
        rotation,
        imageFormat);
Response.BinaryWrite(rawImageBytes);
```

WebFigure

```
// In this example, we use a WebFigure Utility to get an HTML
// String that
// will display this figure, Notice
// how we reference the name we used when attaching it to the
// cache and we indicate
// that the Attach type is session.
String localEmbedString =
    WebFigureServiceUtility.GetHTMLEmbedString(
        "SessionStateWebFigure",
        WebFigureAttachType.session,
        300,
        300);
Response.Write(localEmbedString);
```

Deploying the ASPX

You deploy an ASPX using the **Publish** functionality in Microsoft® Visual Studio. Visual Studio puts all of your code, along with any code your project depends upon, in a folder.

Hosting a DAO Using a .NET Web Service

Setting Up the Root Web Service Class

When creating Web services within .NET, simply create a new Web site (or use an existing site), and add an item of type Web Service to it. This will generate the root class in which you place your methods.

Interacting with the DAO

Each of these methods would be placed in the Web service class as methods.

Numeric

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

String

```
[WebMethod]
public String getString()
{
    Examples.Examples examples = new Examples.Examples();
    String stringResult = examples.getStringFromMWCharArray();
    examples.dispose();
    return stringResult;
}
```

Double Array

```
[WebMethod]
public double[] getDoubleArray()
{
    Examples.Examples examples = new Examples.Examples();
    double[] doubleArray =
        examples.getDoubleArrayFromMWNumericArray();
    examples.dispose();
    return doubleArray;
}
```

Double Matrix

Since .NET Web services can't support multidimensional arrays, convert what is returned from MATLAB Builder NE into a jagged array, as follows:

```
[WebMethod]
public double[][] getDoubleMatrix(int argMagic)
{
    Examples.ExamplesImpl examples =
        new Examples.ExamplesImpl();

    double[,] doubleMatrix =
        examples.getDoubleMatrixFromMWNumericArray(argMagic);

    int arraySize = (int)doubleMatrix.GetUpperBound(0) + 1;
    double[][] outputMatrix = new double[arraySize][];

    int upperOuter = (int)doubleMatrix.GetUpperBound(0) + 1;
    for (int i = 0; i < upperOuter; i++)
    {
        double[] subArray = new double[arraySize];

        int upperInner = (int)doubleMatrix.GetUpperBound(1) + 1;
        for(int j = 0; j < upperInner; j++)
        {
            subArray[j] = doubleMatrix[i, j];
        }
        outputMatrix[i] = subArray;
    }
}
```

```
    }  
  
    examples.dispose();  
    return outputMatrix;  
}
```

Character Array

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

Byte Array

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

Raw Image Bytes

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

Raw Image Bytes with Reorientation

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

Deploying the Web Service


Visual Studio 2005 does all of the work involved with generating Web service artifacts. Once you've created the above methods, just run the service and you'll see a tester page that shows you the location of the WSDL, and then allows you to test each method.

Front-End Developer Tasks

- “Working with the Front-End Layer” on page 5-2
- “Creating a WebFigure on a JSP Page” on page 5-6
- “Creating a WebFigure on an ASPX Page” on page 5-10
- “Working with Static Images” on page 5-13
- “Displaying Complex Data Types Including Arrays and Matrices” on page 5-18
- “Using Web Services” on page 5-26

Working with the Front-End Layer

In this section...
“About the Front-End Layer” on page 5-2
“About the Examples” on page 5-4

 Front-End Developer	Service consumer responsible for presentation and usability No MATLAB experience	Creates front end applications
---	---	--------------------------------

About the Front-End Layer

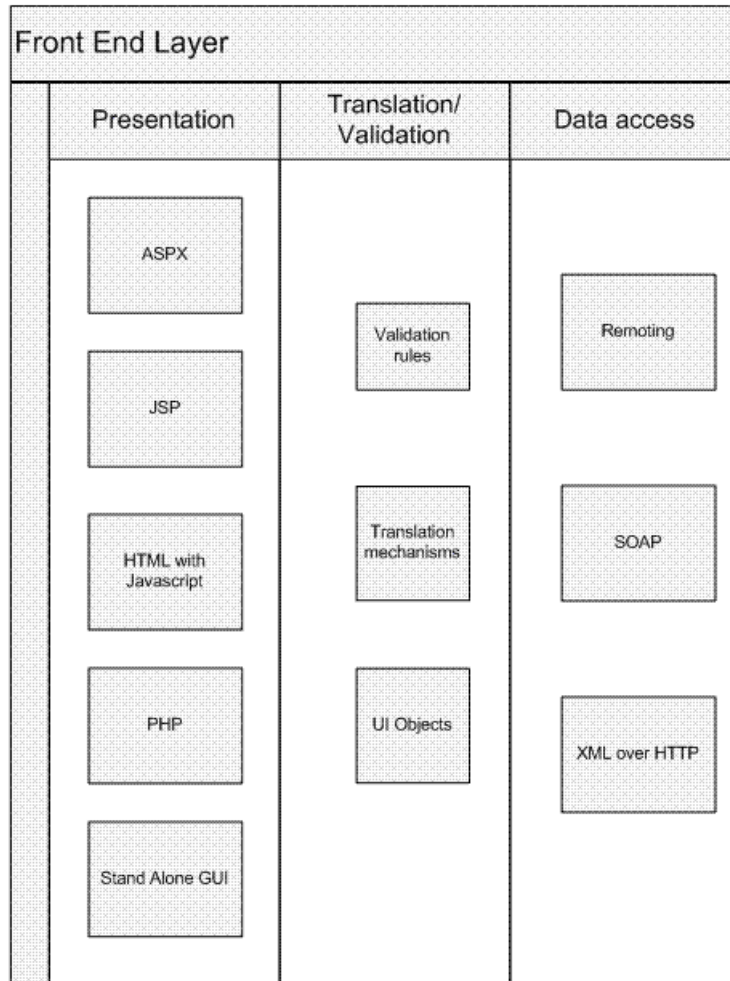
Note For comprehensive end-to-end implementations of the concepts in this chapter, see Appendix A, “Sources for More Information”.

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.



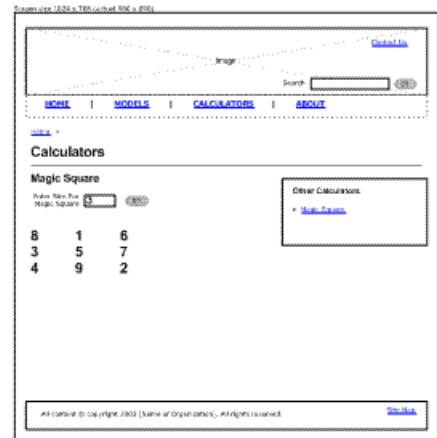
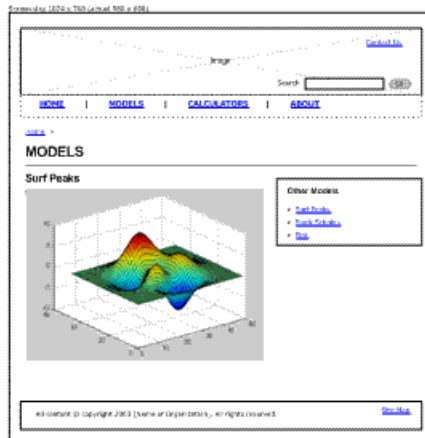
Elements of the Front End Layer

- 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 “Deploying a Java Component Over the Web”, but this example also uses the DAO. The middle tier code in “Hosting the DAO with a Servlet” on page 4-24 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);
```

Creating a WebFigure on an ASPX Page

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

- Using the `WebFigureControl` locally in design mode (the easiest approach)
- Using the `WebFigureControl` remotely in design mode
- Returning an HTML string from the ASPX code behind which embeds the WebFigure into a page.
- Using a `WebFigureControl` directly from the ASPX code without utilizing the designer.

In each case, the `WebFigure` object is stored in the Web server's cache. The JavaScript that executes the client calls back to the server for updates and, using the cached `WebFigure`, new updates are sent back to the client. For more information on these examples, see “Deploying a MATLAB Figure Over the Web Using WebFigures”.

For information on how to create the `WebFigureControl` in Microsoft® Visual Studio®, see “Quick Start: Implementing a WebFigure”.

Using the WebFigureControl Locally in Design Mode

- 1** Drag a `WebFigureControl` from the Microsoft Visual Studio Toolbox to an ASPX page in design mode. This automatically adds a reference to `WebFigureService` to your Web site.
- 2** Edit the code behind the site to attach the figure to the control, as follows:

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

Using the WebFigureControl Remotely in Design Mode

- 1** Drag a `WebFigureControl` from the Microsoft Visual Studio Toolbox onto an ASPX page in design mode.
- 2** Edit the **Properties** for this figure to point it to a Web site that has `webfigureservice` installed and that has attached a `WebFigure` to the

cache. This can either be done using code as shown below or by using the Properties window in the designer.

```
//This is the name that was used on the server
// when attaching the WebFigure
WebFigureControl1.Name = "UserPlot";

//By not setting this value it will try and use
// the current web site for all redirection calls
WebFigureControl1.Root = "";

//If you didn't set this or if it was set to unknown
// this would instruct the WebFigureService to look
// through the diferent caches for a WebFigure with
// this specific name.
WebFigureControl1.Scope = WebFigureAttachType.session;
```

Returning an HTML String from ASPX Code to Embed the WebFigure into a Page.

This example is very similar to what you can find in “Deploying a MATLAB Figure Over the Web Using WebFigures”, but this example also uses the DAO. The middle tier code in “Hosting a .NET DAO with ASPX” on page 4-36 is repeated here:

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

//First we attach the webfigure to one of ASP.NET's caches,
// in this case, the session cache
Session["SessionStateWebFigure"] = webFigure;

//Now we use a WebFigure Utility to get an HTML
// String that will display this figure, Notice
// how we reference the name we used when attaching
// it to the cache and we indicate
// that the Attach type is session.
String localEmbedString =
    WebFigureServiceUtility.GetHTMLEmbedString(
        "SessionStateWebFigure",
        WebFigureAttachType.session,
```

```
        300,  
        300);  
    Response.Write(localEmbedString);
```

Using a WebFigureControl Directly from ASPX Code Without Using the Designer.

To use the WebFigure object directly from an ASPX page, manually write the code that would otherwise be automatically added to the ASPX page. Writing the code creates a WebFigure object with your object's parameters, as in the following example. UserPlot is the name of the Web object that was placed in the cache by the middle tier.

```
<%@ Register Assembly="WebFiguresService, Version=2.8.1.0,  
        Culture=neutral, PublicKeyToken=e1d84a0da19db86f"  
        Namespace="MathWorks.MATLAB.NET.WebFigures.Service"  
        TagPrefix="cc1" %>  
  
<cc1:WebFigureControl  
    ID="WebFigureControl1"  
    runat="server"  
    Height="328px"  
    Width="399px"  
    Root=""  
    Scope="session"  
    Name="UserPlot"  
/>
```

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 its 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:

```

```

Using a Static Image in a ASPX Page

Using ASPX Image objects is almost identical to using HTML `IMG` tags. In this case, simply set the `ImageUrl` (the source of image) to be the ASPX page created in “Deploying the ASPX” on page 4-40. You can also point to a Java middle-tier servlet that hosts the image.

```
<asp:Image
  ID="Image1"
```

```
runat="server"  
ImageUrl=  
    "http://localhost/Examples/Tester.aspx?function=  
        imageBytesFromMWNumeric" />
```

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 4-31 and “Hosting the DAO with a Servlet” on page 4-24. 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%;'>

    </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>");
  }
%>
```

Working with ASPX Page Data

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

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


```

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

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

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

```

Using ASP.NET to Integrate with WYSIWYG Controls

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

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

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

```

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

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

```

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

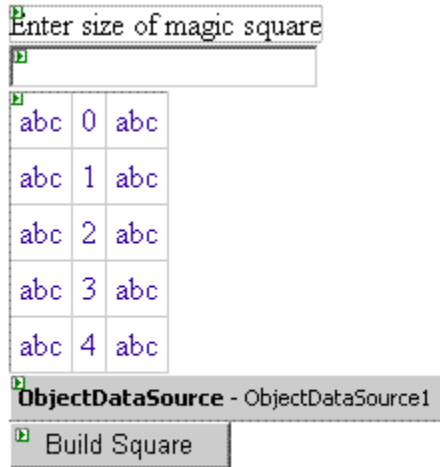
DataRow row;
//Iterate each element in the array creating a row out of each
for(int i = 0; i < size;i++)
{
    //create a row from the table to put the data in
    row = table.NewRow();
    //Iterate each element in the inner array and put
    // them into the row
    for (int j = 0; j < size; j++)
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

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

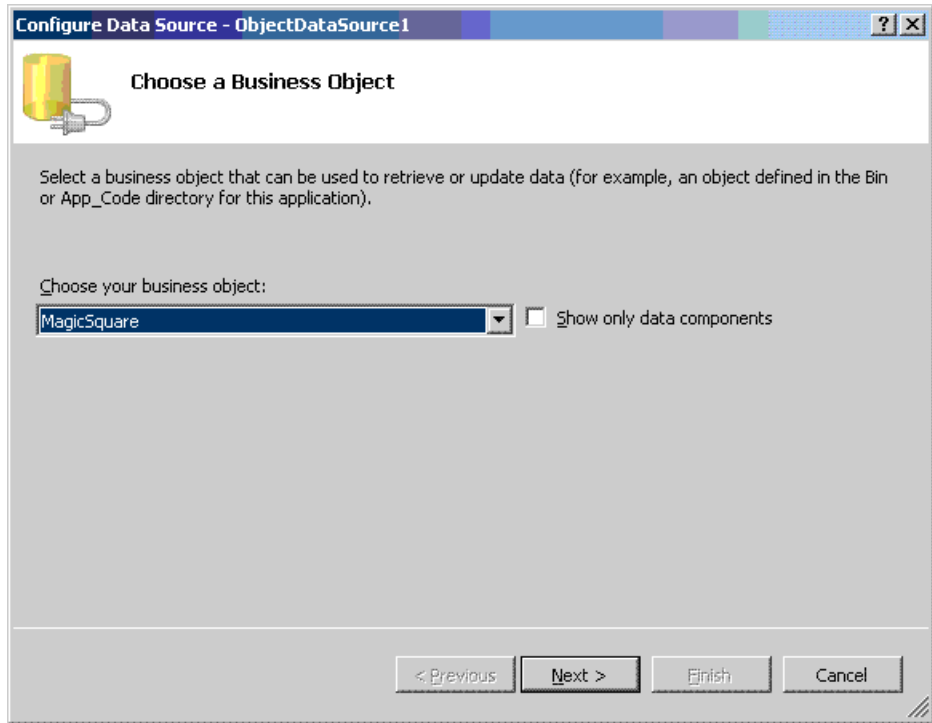
Working with ASP.NET Using the Visual Studio Wizard

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

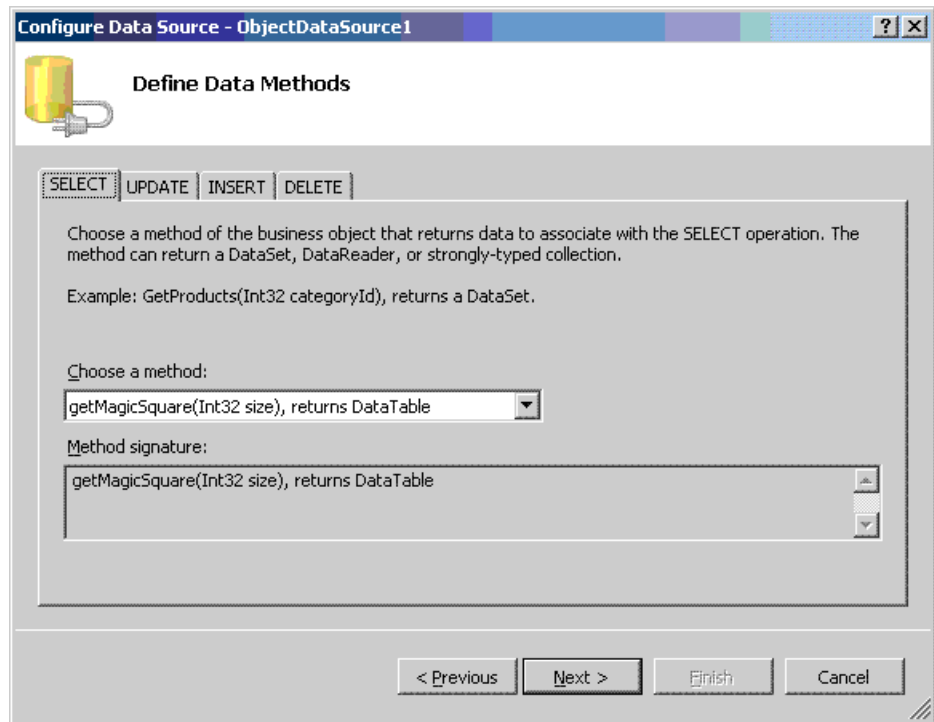
Below is an input text box and a generic grid component from a Web page. The grid component is connected to the ObjectDataSource.



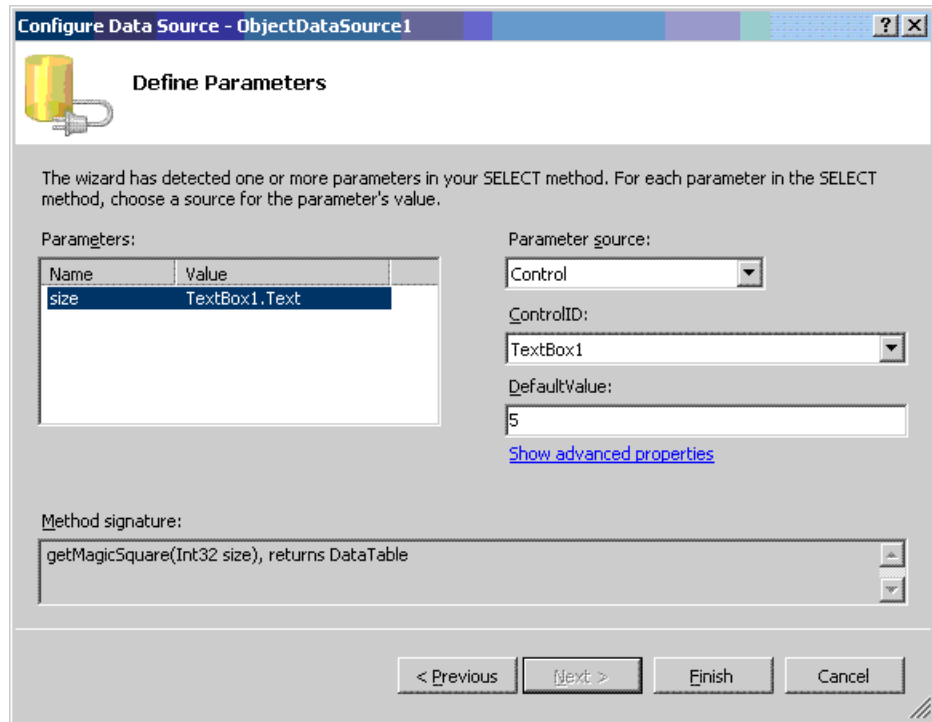
- 1 Start Visual Studio and configure the data source with the wizard.
- 2 Choose the business class that contains your methods for the object.



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



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



The finished application looks like this.

Enter size of magic square

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

Build Square

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

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

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

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

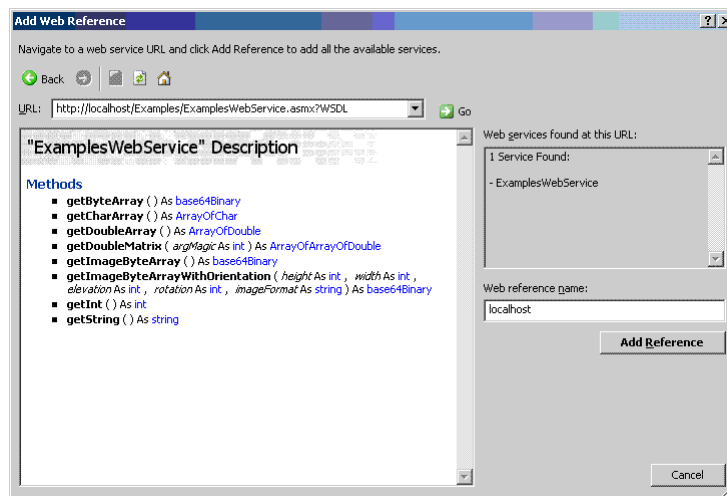
- 1** Install PHP 5.2.3 into IIS 5, if needed (the installer lets you specify the server type).
- 2** Download NUSOAP and place it on the instance path.

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

Displaying Web Services Images and Data in .NET

In Chapter 4, “Business Service Developer Tasks”, you exposed several methods as SOAP Web services. To use these methods from the front end, add a reference to them in your Visual Studio project by adding a **Web Reference**:

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



Notice how all of the methods exposed earlier are displayed.

- 4 To add these methods to your project, click **Add Reference**.
- 5 To use any of the methods, instantiate the Web service by executing code similar to:

```
localhost.ExamplesWebService webService =  
new localhost.ExamplesWebService();
```

- 6 Access any of the methods on the Web service as you would any other .NET object. Here are some examples:

```
int intValue = webService.getInt();  
string stringValue = webService.getString();  
double[] doubleArray = webService.getDoubleArray();  
double[][] magicSquare = webService.getDoubleMatrix(size);  
char[] charArray = webService.getCharArray();  
byte[] byteArray = webService.getByteArray();  
byte[] imageByteArray = webService.getImageByteArray();  
byte[] imageByteArrayWithOrientation =  
webService.getImageByteArrayWithOrientation( 500,  
                                               500, 20, 30, "png");
```

Server Administrator Tasks

- “Managing a Deployment Server Environment” on page 6-2
- “Hot Deployment” on page 6-11
- “Working with Multiple Versions of the MATLAB Compiler Runtime (MCR)” on page 6-12
- “Unsupported Versions of the JVM” on page 6-13

Managing a Deployment Server Environment

In this section...

“The Server Administrator’s Role in Deployment” on page 6-2

“An Overview of Deployed Applications” on page 6-2

“Installing the MATLAB Compiler Runtime (MCR)” on page 6-3

“Loading the MATLAB Compiler Runtime (MCR)” on page 6-6

“Scaling Your Server Environment” on page 6-8

“Ensuring Fault Tolerance” on page 6-9

 Server Administrator	IT Expert Understands server architectures Understands .NET frameworks and Java environments	Responsible for up-time of production servers; meeting SLA requirements
---	---	---

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.

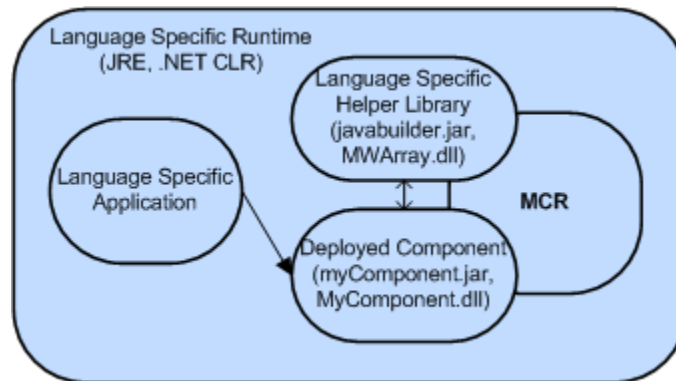
System management requires ensuring that one set of applications does not affect another and also managing version conflicts that arise. Another concern of the server administrator is monitoring system up-time. The following sections discuss the nuances of the deployment products and what the server 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 *MCR*. 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 MCR 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 MCR (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 Compiler Runtime (MCR)

Because the MCR provides the run-time infrastructure that runs deployed applications, it 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 MCR, you include references to specific directories on your Windows®, Linux®, or Mac OS® X path. For more information, see “Install the MCR on Target Computers Without MATLAB and Update System Paths”.

The MATLAB Compiler Runtime (MCR) Installer

The MCR installer ships with MATLAB. You obtain it from the MATLAB programmer who originally compiled the application. The version number of the MCR 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 MCR. This library contains information for communicating with the MCR. It also communicates with helper utilities and data types implemented by the wrapper code. Using these wrappers, you can convert your data to and from MATLAB data types.

.NET

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

- `mcrroot\toolbox\dotnetbuilder\bin\win32|win64\vmajor.minor\MWArray.dll`
— Contains MATLAB data type wrappers, utilities, and MCR communication utilities
- `mcrroot\toolbox\dotnetbuilder\bin\win32|win64\vmajor.minor\WebFiguresService.dll` — Contains `WebFiguresService`, `WebFigureControl`, , and `WebFigure` rendering utilities)

Note `vmajor.minor` denotes the version of the .NET framework. See the “Product Overview” in the MATLAB Builder NE documentation for a list of supported framework versions.

Note *mcrroot* is the MCR's installation directory.

Both of these libraries install into the *Global Assembly Class (GAC)* automatically when you install the MCR. Although they reside in the GAC, you reference them from Microsoft Visual Studio during development. If you are using the `WebFigureControl` from within Visual Studio, drag the `WebFiguresService.dll` into the toolbox.

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

Java

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

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

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

For a Web server, place this component in the shared library location for your server to allow all Web applications to inherit these libraries. In older versions of Tomcat, this directory is `tomcatroot/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 MCR's installation directory.

Loading the MATLAB Compiler Runtime (MCR)

Because deployed applications use the MCR at run-time, the MCR must be loaded and running. The MCR loads when a class in a deployed component is instantiated for the first time.

MCR loading can take anywhere from 10 seconds to over a minute (if the MCR resides on a network). To ensure a consistent user experience for Web applications, load the MCR at server start-up time, rather than upon first use, by instantiating a class as part of the server initialization.

.NET

In ASP.NET Web applications, create a Global Application Class (Global.asax) using this code:

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

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

For more information about the Global.asax file, see “Using Global Application Class (Global.asax) to Create WebFigures at Server Start-Up”.

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

Java

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

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


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

2 Place this code in your class:

```
import javax.servlet.*;

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

        //Instantiate the deployed component,
        // this will trigger the 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 6-8
- “Session Scaling” on page 6-8

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 cluster 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 6-8 to optimize performance.

Session Scaling

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

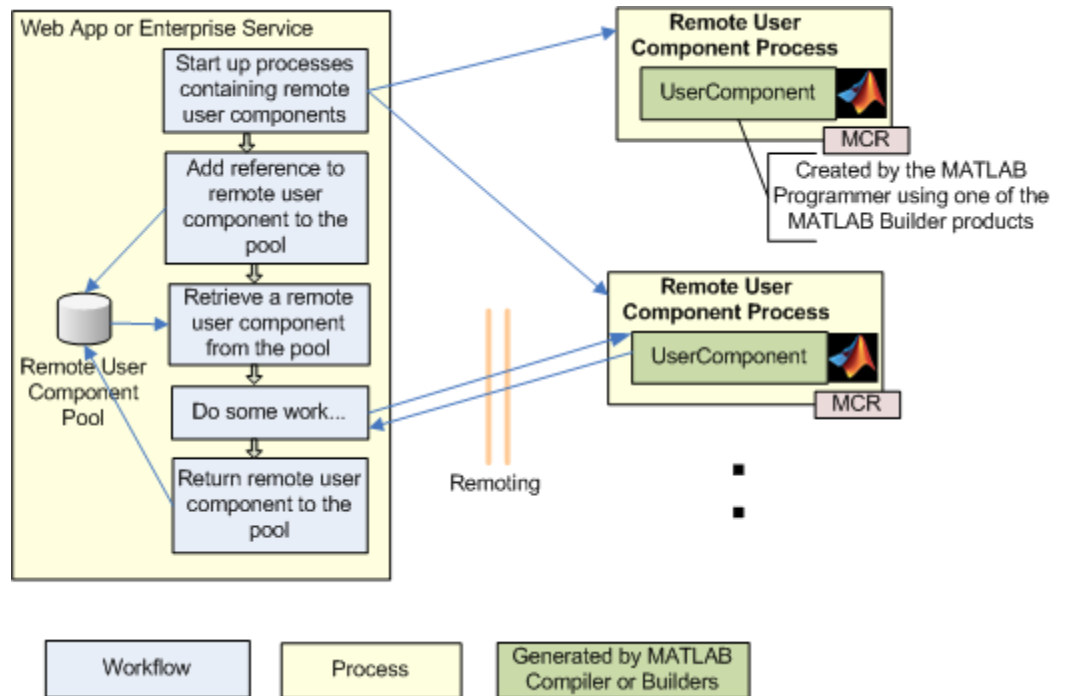
Many session scaling issues arise because the MCR is single threaded. A single-threaded application prevents two users from doing work that involves the MCR at the same time. One user must wait for the other to finish before continuing. This wait can prove to be substantial if one user is performing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

To workaroud the situation, enable multiple MCRs to service requests as they arrive. Run several MCRs in separate processes; one process per MCR. 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 .NET Remoting 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 MCR. As your usage grows, add in a scaling layer as needed. Adding another layer involves minimal client changes.



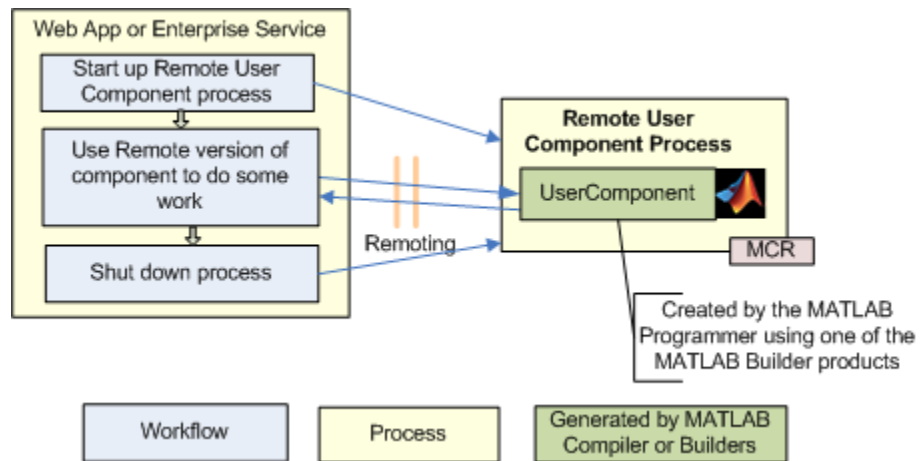
Remote Scaling

Ensuring Fault Tolerance

In your default scenario, avoid using the MCR in the same process as the rest of your application. The MCR is a large complex native application running inside your applications process. If the MCR runs 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 6-8. Remoting allows you to start up a process whose only job is to start the MCR 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 an MCR 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.

.NET

MATLAB Builder NE supports Hot Deployment of deployed components in either:

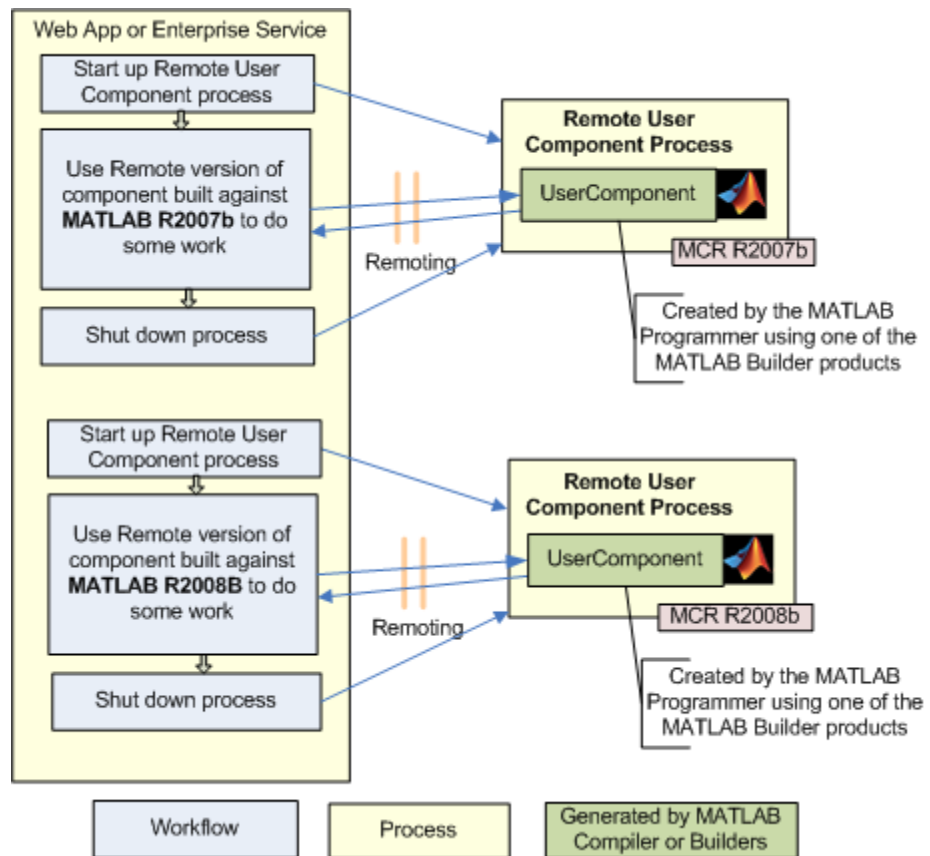
- The WebDev server (the version of IIS built into Microsoft Visual Studio)
- IIS itself

Java

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

Working with Multiple Versions of the MATLAB Compiler Runtime (MCR)

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 MCR Libraries loads into any given process.

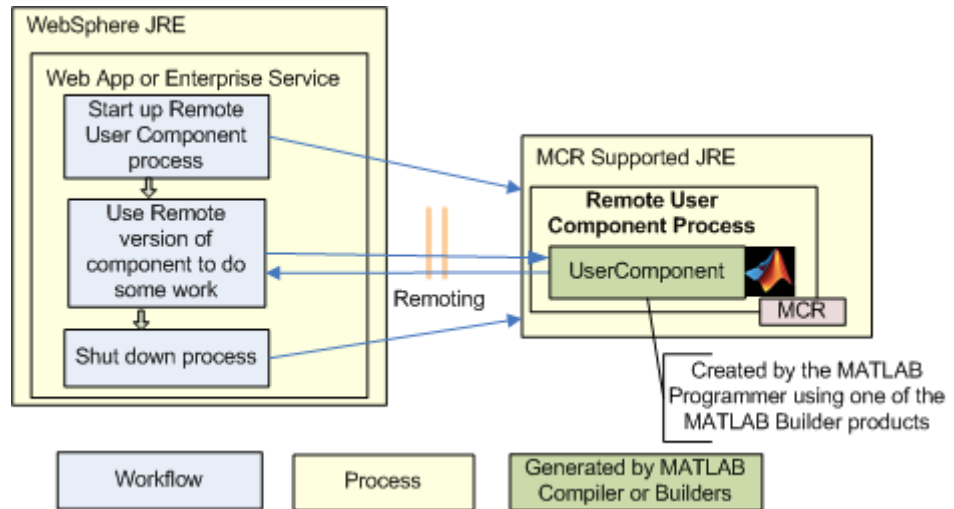


Using Remoting to Workaround Multiple Versions of the MCR

Unsupported Versions of the JVM

The MCR 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 MCR 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 Sun™ JVM, for example. You can workaround this issue by using remoting to pull the MCR into its own process, where it uses the proper JVM.




Using Remoting to Workaround an Unsupported JVM

Internal Analyst Tasks

- “Working with Content” on page 7-2
- “Example Tasks” on page 7-3

Working with Content

 Internal Analyst	Internal user over the network Little to no MATLAB or IT experience	Employer of the company User of tools and sites
--	--	--

Internal analysts 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.

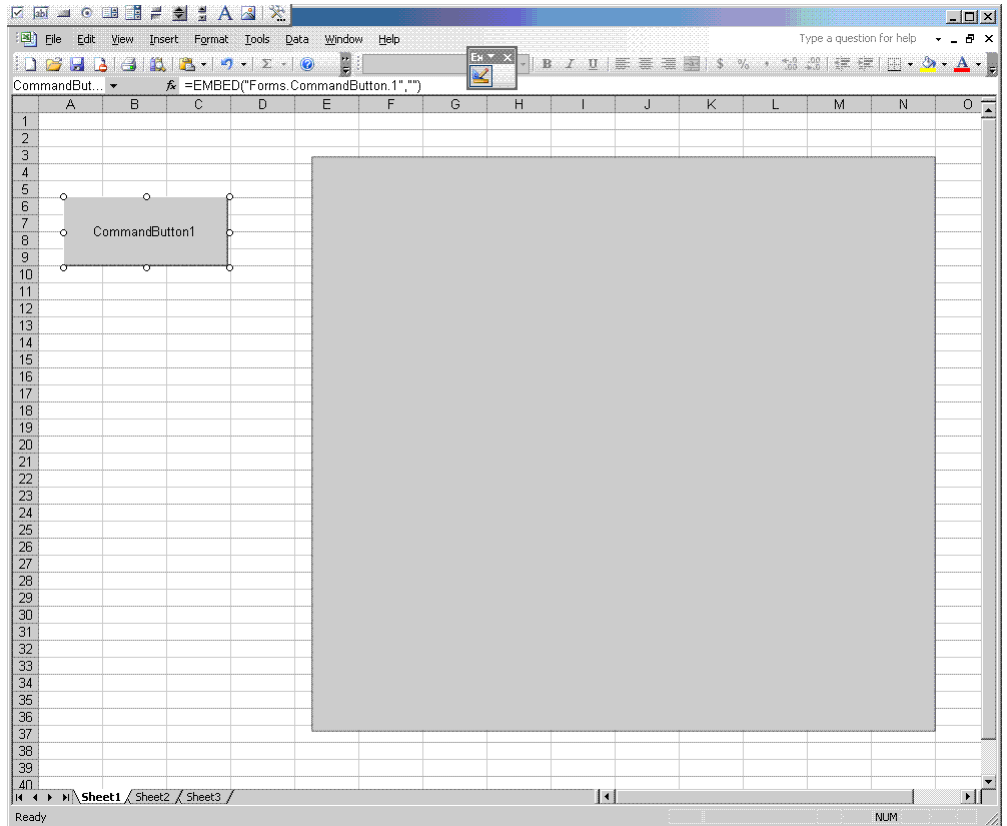
Example Tasks

The example in this section consumes the Web service created in “Deploying the Web Service” on page 4-34. 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–riven 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.

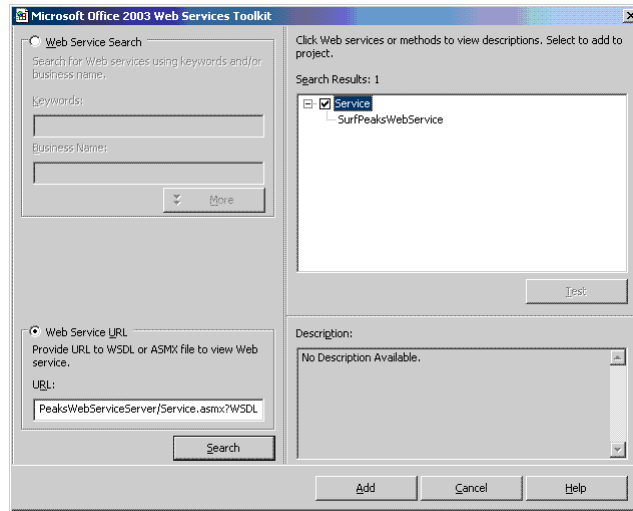
7 Select **Tools > Web Service References**.

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 5-26:

<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

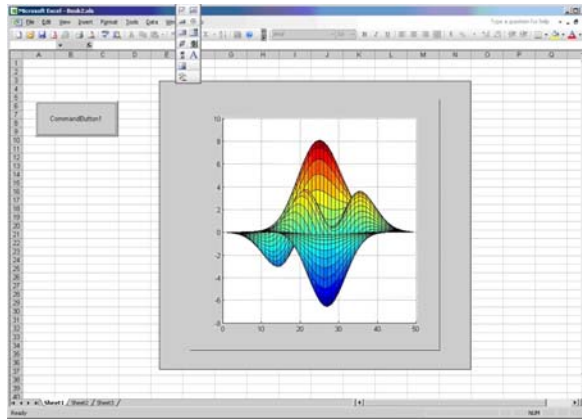
Dim value() As Byte
Set module = New clsWS_Service
value = module.wsm_SurfPeaksWebService

'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.



End-to-End Developer Tasks

- “Role of the End-To-End Developer” on page 8-2
- “Example: The Magic Square Calculator On the Web” on page 8-3
- “Creating an End-to-End Web Application” on page 8-5

Role of the End-To-End Developer

 End to End Developer	MATLAB Expert Creates services out of components received from MATLAB Programmer Service consumer responsible for presentation and usability	Develops model Uses builder tools to create a component given to the business service developer Creates scalable services and apps
---	--	--

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.

Example: The Magic Square Calculator On the Web

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

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

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

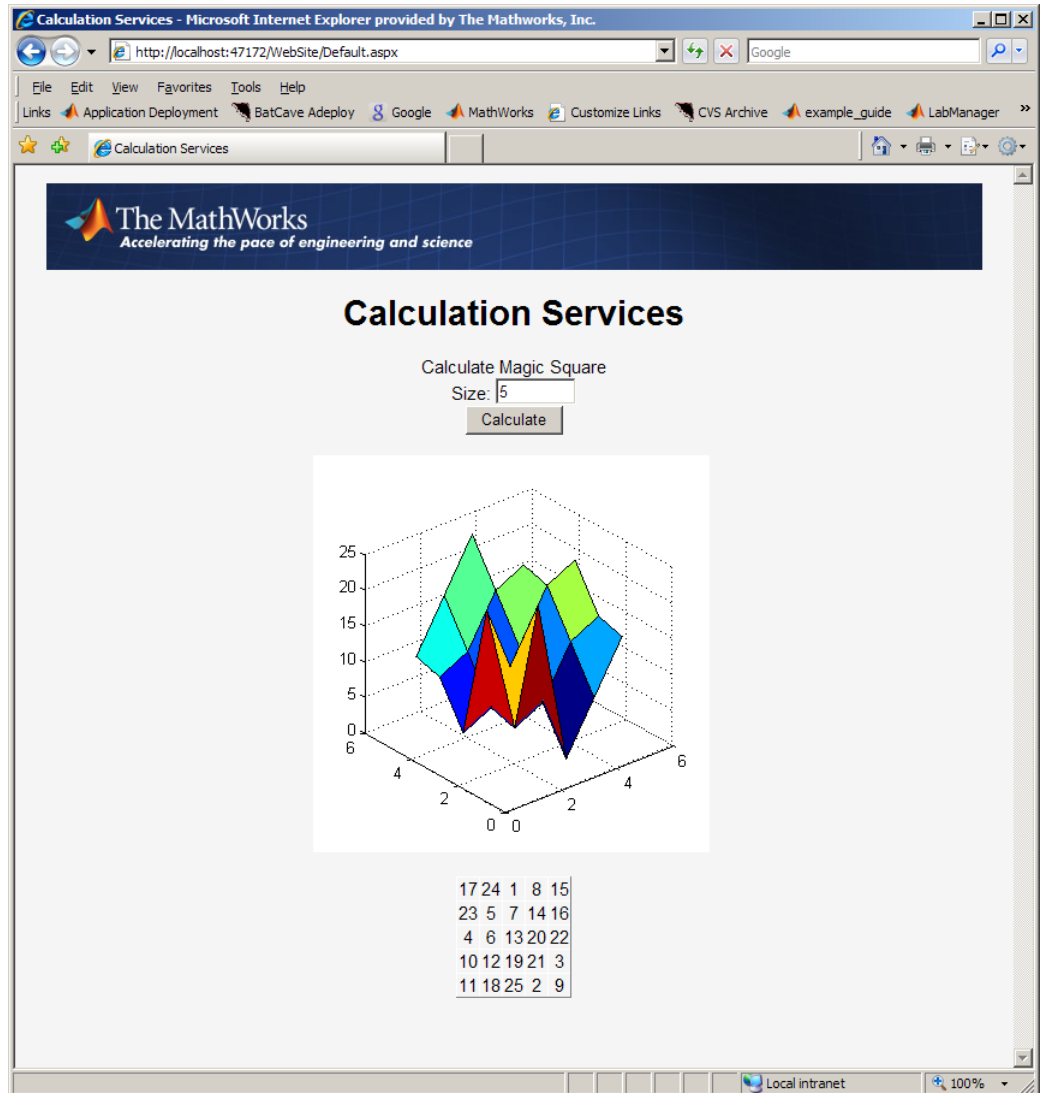
getmagic.m:

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

getMagicWebFigure.m:

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

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



Magic Square Calculator Application Presented on the Web

Creating an End-to-End Web Application

In this section...

“Creating a Java Web Application, End-to-End” on page 8-5

“Creating a .NET Web Application, End-to-End” on page 8-13

Creating a Java Web Application, End-to-End

In order to deploy the Magic Square Calculator application described in “Example: The 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 “Quick Start: Building Your Example Component” on page 8-5.

Quick Start: 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`.

Advanced: 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 MCR. 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 MCR you have installed is the same version as the MCR running with MATLAB when the application was built. If you are unsure, check with your MATLAB programmer or whoever initially deployed the component. For more information, see "Deploying a Component with the Magic Square Example" in the MATLAB Builder JA User's Guide.
- 4 Make note of the folder where the MCR 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 "Quick Start: 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></td>
          </tr>
        </table>
        <br />

        <h1> Calculation Services</h1>

        Calculate Magic Square
        <br>
        Size:
        <input type="text" name="size" size="8" value="<%=size%>" >
        <br>
      </div>
    </form>
  </body>
</html>
```

```
<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 XMLHttpRequest();
        }
        catch (e)
        {
            try
            {
                var objXHR = new XMLHttpRequest();
            }
            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 `MCRROOT/toolbox/javabuilder/jar/javabuilder.jar`) to the `webapp/WEB-INF/lib` folder.
- 8** Since you are compiling a servlet which has J2EE dependencies, copy `servlet-api.jar` (the J2EE JAR file that comes with Tomcat) to the current working folder. This file is usually located in the Tomcat common `lib` folder (a J2EE JAR file can work as well).
- 9** From the root of your `Projects` folder we need to build the component with MATLAB by using the following `mcc` command:

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

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

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

- 11** Rebuild the servlet by typing the following command:

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

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

12 To rebuild the WAR file, enter:

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

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

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

- 1** The MATLAB Builder NE DLL file that is generated by compiling MATLAB code using MATLAB, MATLAB Compiler, and MATLAB Builder NE.
- 2** The DataTable implementation that converts the magic square output into something that an ObjectDataSource can use in a GridView Control.
- 3** The ASPX page and the code behind it. This page is responsible for taking in user input, displaying the controls, and handling page events.

Preparing Your Example Application

- 1** Ensure you have a supported .NET Framework installed and Visual Studio 2005 or later. See the “Product Overview” in the MATLAB Builder NE documentation for a list of supported framework versions.
- 2** Ensure the version of the MCR you have installed is the same version as the MCR running with MATLAB when the application was built. If you are unsure, check with your MATLAB programmer or whoever initially deployed the components. See the MATLAB Compiler User’s Guide reference pages for details on the `mcrversion` command.

- 3 Make note of the folder where the MCR is installed. It will be used later when starting the applications.
- 4 Create the code for the DataTable implementation:

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

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

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

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

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

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

```

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

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

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

```

5 Create the code for the ASPX page:

```

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

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

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

```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head id="Head1" runat="server">
    <title>Calculation Services</title>
    <link rel="stylesheet" type="text/css"
      media=all href="./StyleSheet.css" />
    <link href="StyleSheet.css"
      rel="stylesheet" type="text/css" />
  </head>
  <body>
    <form id="form1" runat="server">
  <div style="text-align: center">
    <table width="760" cellpadding="0" cellspacing="0">
      <tr>
        <td>
  </td>
      </tr>
    </table>
    <br />
    <h1> Calculation Services</h1>
    <asp:Label
  ID="Label3"
  runat="server"
  Text="Calculate Magic Square">
  </asp:Label>
  <br />
    <asp:Label
  ID="Label4"
  runat="server"
  Text="Size: ">
  </asp:Label>
  <asp:TextBox
  ID="TextBox3"
  runat="server">
```

```
Width="61px">
5
</asp:TextBox>
<br />
<asp:Button
ID="Button2"
runat="server"
OnClick="Button2_Click"
Text="Calculate" />
<br />
<br />
<cc1:WebFigureControl
ID="WebFigureControl1"
runat="server"
Height="330px"
Width="330px" />
<br />
<br />
<asp:ObjectDataSource
ID="ObjectDataSource1"
runat="server"
SelectMethod="getMagicSquare"
TypeName="MagicSquare">
<SelectParameters>
<asp:ControlParameter
ControlID="TextBox3"
DefaultValue="5"
Name="size"
PropertyName="Text"
Type="Int32" />
</SelectParameters>
</asp:ObjectDataSource>
<asp:GridView
ID="GridView1"
runat="server"
```

```
                DataSourceID="ObjectDataSource1"  
                ShowHeader="False">  
            </asp:GridView>  
        </div>  
    </form>  
</body>  
</html>
```

6 Create the code behind the ASPX page:

```
using System;  
using System.Data;  
using System.Configuration;  
using System.Web;  
using System.Web.Security;  
using System.Web.UI;  
using System.Web.UI.WebControls;  
using System.Web.UI.WebControls.WebParts;  
using System.Web.UI.HtmlControls;  
using Examples;  
using MathWorks.MATLAB.NET.WebFigures;  
using MathWorks.MATLAB.NET.Arrays;  
  
public partial class _Default : System.Web.UI.Page  
{  
    private MagicCalc calc = new MagicCalc();  
    protected void Page_Load(object sender, EventArgs e)  
    {  
        MWArray input = Int32.Parse(TextBox3.Text);  
        WebFigureControl1.WebFigure =  
            new WebFigure(calc.getMagicWebFigure(input));  
    }  
    protected void Button2_Click(object sender, EventArgs e)  
    {  
        MWArray input = Int32.Parse(TextBox3.Text);  
        WebFigureControl1.WebFigure =  
            new WebFigure(calc.getMagicWebFigure(input));  
  
        ObjectDataSource1.Select();  
    }  
}
```



```
}
```

Building Your Example Application

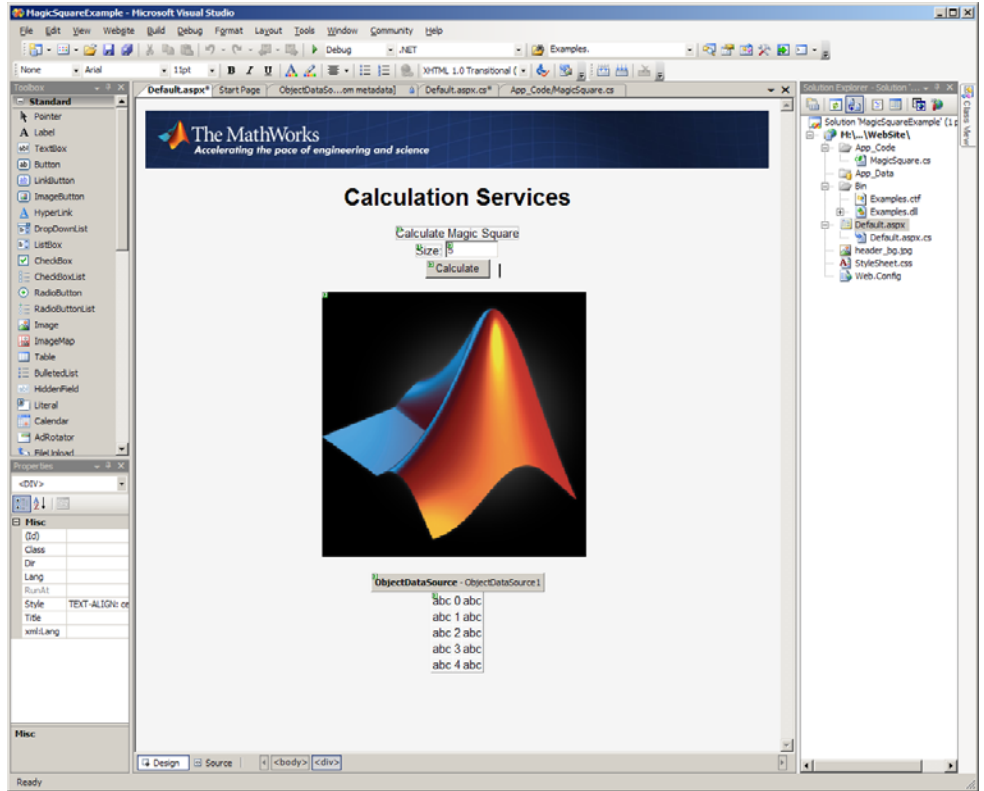
Build your application and deploy it using the version of IIS built into Microsoft Visual Studio as follows:

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

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

This command creates a file in the `WebSite\mcode\output` folder called `Examples.dll` (this will be referenced from the Visual Studio project in the next step).

- 3 Open Microsoft Visual Studio 2005 and select **File > Open > Web Site**.
- 4 Browse to the `WebSite` folder and open it.
- 5 Confirm that `Examples.dll` is added correctly as a reference. If there were any errors or issues adding it, delete it and add a new reference to the DLL you built in the `WebSite\mcode\output` folder.



Microsoft® Visual Studio® Designer View of the Magic Square Calculator

- 6** Select **Build > Build Solution**. You should build with no errors. If there are errors, ensure that:
 - The MCR is installed.
 - Examples.dll has been added as a reference in your project.
- 7** When you have built successfully, select **Debug > Start Debugging**. A local IIS server starts and opens your page inside of it.

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 demo for Java

WebFigures

MATLAB Builder JA WebFigures Varargs demo

MATLAB Builder NE

Other examples using MATLAB Builder NE include:

Black-Scholes

MATLAB Central Black-Scholes Web demo for .NET

B

business service layer 4-2

D

DAO 4-4

 Hosting

 Using a .NET Web Service 4-40

 Using a Java Web Service 4-31

 Hosting On a Web Server 4-24

 Initializing

 For a Servlet 4-25

 Initializing the

 with ASPX 4-36

 Interacting with 4-37 4-40

 in Java 4-32

 Using an HTML String 5-6

F

Fault Tolerance

 ensuring 6-9

front end layer 5-2

H

Hot Deployment 6-11

J

Java Virtual Machine (JVM)

 Workaround for unsupported JVMs 6-13

M

MATLAB Compiler Runtime (MCR)

 Workaround for multiple MCR versions 6-12

N

NUSOAP

 Download location 5-26

NUSOAP PHP class add-in 1-5

S

Scaling

 Calculation method (using PCT) 6-8

 Session method 6-8

Server Administrator

 Role 6-2

Server Environment

 Scaling 6-8

SOAP 4-31

SOAP Web services

 As part of data access in front-end layer 5-3

 As part of interfaces in business service
 layer 4-2

 Using Embedded HTML with 5-8

 Using To embed an image on a PHP
 page 5-26

 Using Web references with 5-27

SOAP Web Services 1-2

W

Web Figure

 WebFigure 2-19

WebFigures 4-5

 and MATLAB figures 3-3

 Deploying a Java component 4-28

 Example

 using a DAO 4-28

 Getting image data from a WebFigure 4-23

 JSP page 5-6

 Other examples of A-2

 Returning a figure as data 3-4

 Returning data from a WebFigure

 Window 3-3

 Use in AJAX component 5-5

 Using directly from JSP

Using tag library 5-7