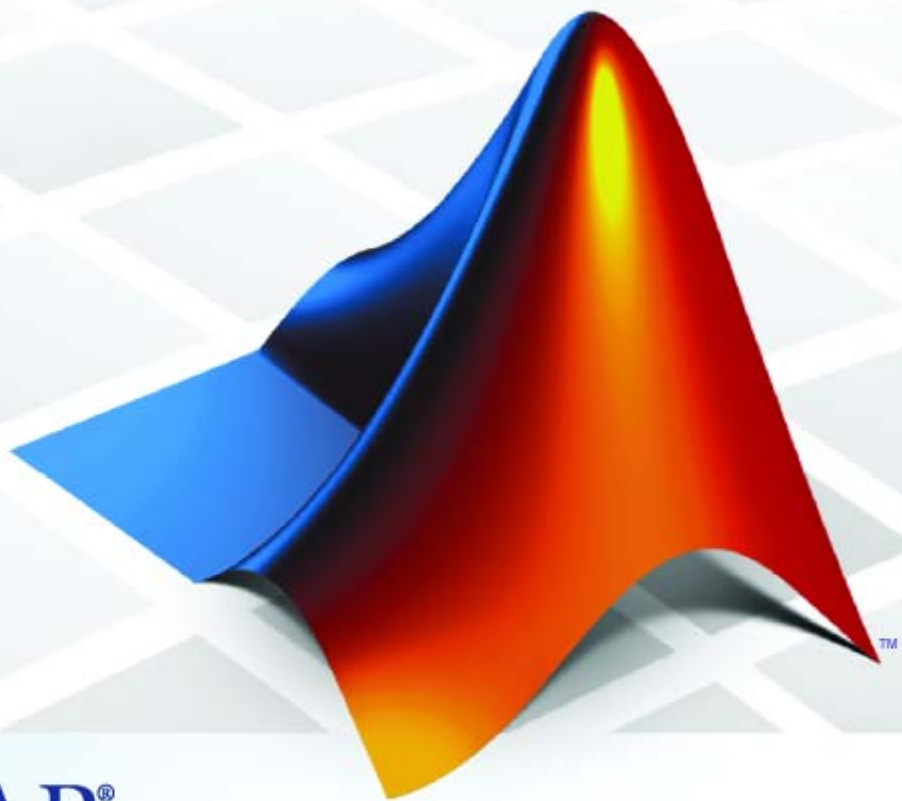


# MATLAB® Application Deployment Web Example Guide



MATLAB®

## How to Contact The MathWorks



[www.mathworks.com](http://www.mathworks.com)  
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab)  
[www.mathworks.com/contact\\_TS.html](http://www.mathworks.com/contact_TS.html)

Web  
Newsgroup  
Technical Support



[suggest@mathworks.com](mailto:suggest@mathworks.com)  
[bugs@mathworks.com](mailto:bugs@mathworks.com)  
[doc@mathworks.com](mailto:doc@mathworks.com)  
[service@mathworks.com](mailto:service@mathworks.com)  
[info@mathworks.com](mailto:info@mathworks.com)

Product enhancement suggestions  
Bug reports  
Documentation error reports  
Order status, license renewals, passcodes  
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 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used 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.

**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](http://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](http://www.mathworks.com/patents) for more information.

### Revision History

March 2008      Online only      First edition

## How to Use This Guide

### 1

<b>About This Guide</b> .....	1-2
<b>Who Should Use This Guide?</b> .....	1-3
<b>List of Commonly Used Software and Preliminary Set-Up Information</b> .....	
<b>Set-Up Information</b> .....	1-4
<b>MATLAB Programmer</b> .....	1-4
<b>Integration Experts (Business Service Developer and Front-End Developer)</b> .....	1-4
<b>Internal Analyst</b> .....	1-5

## Anatomy of a MATLAB Web Application

### 2

<b>The MATLAB Web Application Environment</b> .....	2-2
<b>The Lifecycle Of a Deployed MATLAB Web Application</b> .....	
<b>The MATLAB Programmer</b> .....	2-4
<b>The Business Service Developer</b> .....	2-6
<b>The Front-End Developer</b> .....	2-7
<b>The Internal Analyst</b> .....	2-7

## MATLAB Programmer Tasks

### 3

<b>Programming in MATLAB</b> .....	3-2
<b>Returning Data Types</b> .....	3-3

## Business Service Developer Tasks

### 4

<b>Working with the Business Service Layer</b> .....	<b>4-2</b>
About the Business Service Layer .....	<b>4-2</b>
Creating a DAO For Deployment .....	<b>4-5</b>
Hosting a DAO On a Web Server .....	<b>4-20</b>

## Front End Developer Tasks

### 5

<b>Working With the Front End Layer</b> .....	<b>5-2</b>
Creating a Web Figure On a JSP Page .....	<b>5-5</b>
Working With Static Images .....	<b>5-8</b>
Displaying Complex Data Types Including Arrays and Matrices .....	<b>5-12</b>
Using Web Services .....	<b>5-19</b>

## Internal Analyst Tasks

### 6

<b>Working With the Content</b> .....	<b>6-2</b>
<b>Examples of Internal Analyst Tasks</b> .....	<b>6-3</b>

## Sources for More Information

---

**A**

<b>Other Examples</b> .....	<b>A-2</b>
MATLAB® Builder™ JA .....	<b>A-2</b>
MATLAB® Builder™ NE .....	<b>A-2</b>

**Index**

---



# How to Use This Guide

---

About This Guide (p. 1-2)	A brief description of the Example Guide and how it can help you
Who Should Use This Guide? (p. 1-3)	A brief description of the skill sets of potential users of this guide, and recommended reading paths
List of Commonly Used Software and Preliminary Set-Up Information (p. 1-4)	Software typically needed when using this guide and associated preliminary set-up information

## About This Guide

The MATLAB® Compiler™ and builders (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.

## List of Commonly Used Software and Preliminary Set-Up Information

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

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

### **MATLAB Programmer**

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







### **Intergration 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 2.0 or later
- 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 pre-processor 5.2.3 or later
- NUSOAP PHP class add-in

## Internal Analyst

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

		Business	Integration Experts		End User		End-to-end User
Roles		 <b>MATLAB Programmer</b>	 <b>Business Service Developer</b>	 <b>Front End Developer</b>	 <b>External User</b>	 <b>Internal Analyst</b>	 <b>General Developer</b>
	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)
		Java Builder MATLAB, Compiler .NET Builder	MCR, IIS, Soap server (IIS)	IIS, home grown apps (e.g., Excel), third party apps (e.g., Siebel)		Excel	MCR, IIS, Soap server (IIS), home grown apps (e.g., Excel), third party apps (e.g., Siebel)



# Anatomy of a MATLAB Web Application

---

The MATLAB Web Application Environment (p. 2-2)

An overview of the deployment environment and tools used to work within it

The Lifecycle Of a Deployed MATLAB Web Application (p. 2-4)







How M-code becomes a deployable language-agnostic enterprise application

### **The 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), 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.

Web Figures is a client and server side 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 Web Figures functionality of MATLAB Builder JA allows users limited to Web access the ability to dynamically interact with MATLAB figures.

	Role	Knowledge Base	Responsibilities
Business	 <b>MATLAB Programmer</b>	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	 <b>Business Service Developer</b>	No MATLAB experience Creates services out of components received from the MATLAB programmer	Creates reusable service that is scalable
	 <b>Front End Developer</b>	Service consumer responsible for presentation and usability No MATLAB experience	Creates front end applications
End User	 <b>External User</b>	External user over the internet through firewalls No MATLAB or IT experience	Customer of the company User of the site
	 <b>Internal Analyst</b>	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	 <b>General Developer</b>	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 Web Application Deployment Roles**

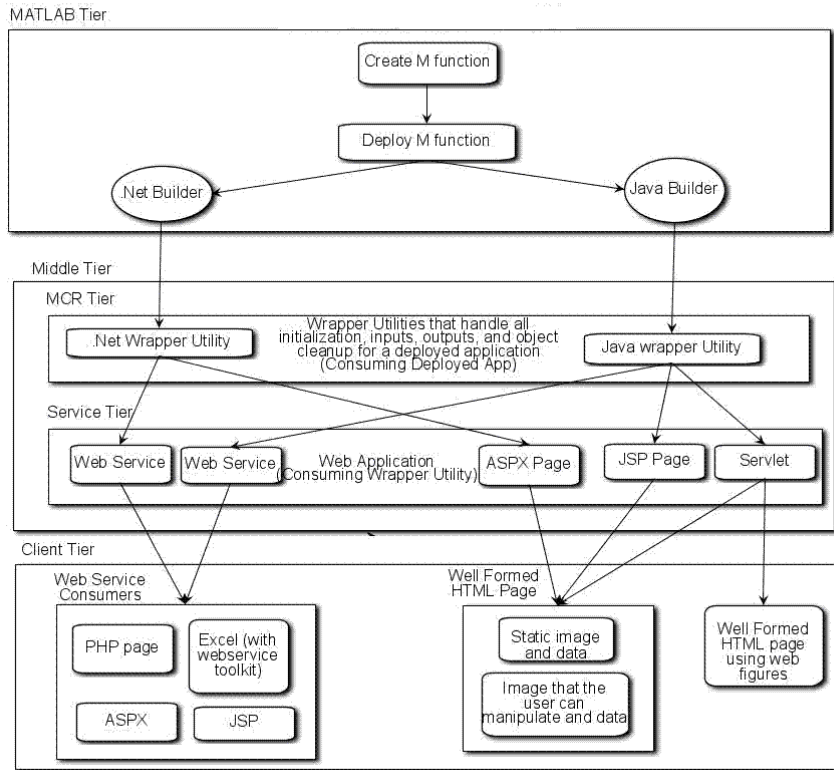
## The Lifecycle Of a Deployed MATLAB Web Application

In this section...
“The MATLAB Programmer” on page 2-6
“The Business Service Developer” on page 2-7
“The Front-End Developer” on page 2-7
“The Internal Analyst” on page 2-7

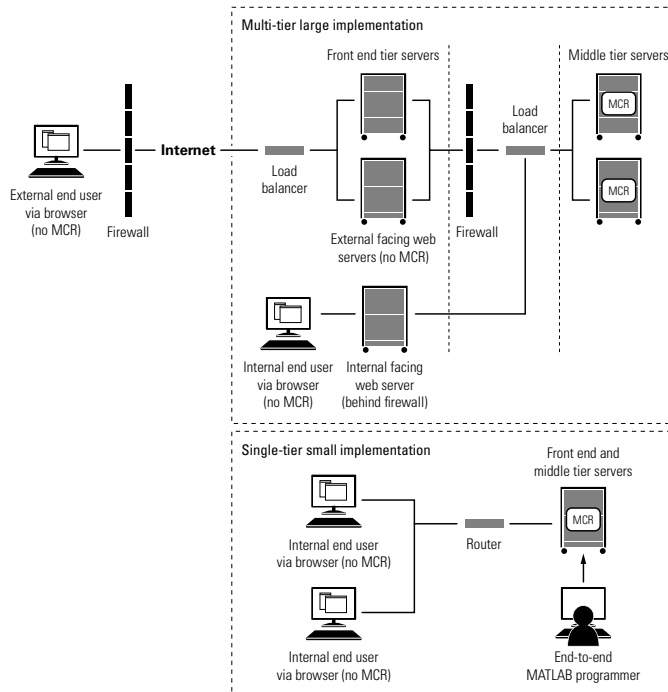
How does a single piece of M-code become a deployable, portable, robust, scalable Web application? Through skillful deployment by a number of people in an organization, each playing distinct and significant roles.

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





**Deployed Application Tier Interactions**



### The MATLAB Programmer

The first phase in a deployed application's life begins when code is written in MATLAB by a MATLAB Programmer, who has the objective in mind of sharing it with other people either within or outside of his organization. To accomplish this objective, the programmer uses MATLAB Compiler. MATLAB Compiler makes M-code usable by people in vastly different environments who may not have knowledge of MATLAB or the M language.

When MATLAB Builder JA (for Java language) or MATLAB Builder NE (for Microsoft .NET framework), are installed along with MATLAB Compiler, M functions can be encrypted and wrapped in Java or .NET interfaces. The MATLAB Programmer takes these deployable components and hands them off to the Business Service Developer.

## **The 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.

## **The 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 Developer creates 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.

## **The 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 Excel model. Or, they access an internal Web site, performing specific tasks not available to their customers.



# MATLAB Programmer Tasks

---


Programming in MATLAB (p. 3-2)

Examples of tasks to perform while programming in MATLAB

Deploying MATLAB Code With the Builders (p. 3-6)

How to deploy your MATLAB code using the MATLAB builders

## Programming in MATLAB

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

MATLAB is a interpreted programming environment. Functions can be executed 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 M-code within functions. MATLAB does not allow inline scripts to be used.

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. In later parts of this guide, we 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 can be created in MATLAB can be deployed, providing it is contained in a function. For example,

```
>> 1 + 1
```

can not 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. Any primitive data type can be used 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 in-depth explanation of how to work with MATLAB primitive data types, see *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 Web Figures infrastructure (see “Deploying a Java™ Component Over the Web”), MATLAB Builder JA 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 Web Figure Window**

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

In the following example, the figure should be closed 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 Web Figures. If the figure is attached to the rest of the infrastructure, it will automatically pass, resize, and reorient accordingly.

```
%returns a web figure reference containing the
%data from the figure window
function resultWebFigure = getWebFigure
    f = figure;
    surf(peaks);
    resultWebFigure = webfigure(f);
    close(f);
end
```

### **Returning a Figure as Data**

This approach is typically used for instances where Web figures 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 M-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 M-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. Typically the best practice is to ensure the M-files that contain a method have the same name as the M-file for all entry point methods.

For examples using MATLAB's 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 (p. 4-2)

Examples of tasks to perform while  
at the business service layer

# Working with the Business Service Layer

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

### In this section...

“About the Business Service Layer” on page 4-2

“Creating a DAO For Deployment” on page 4-5

“Hosting a DAO On a Web Server” on page 4-20

---

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

---

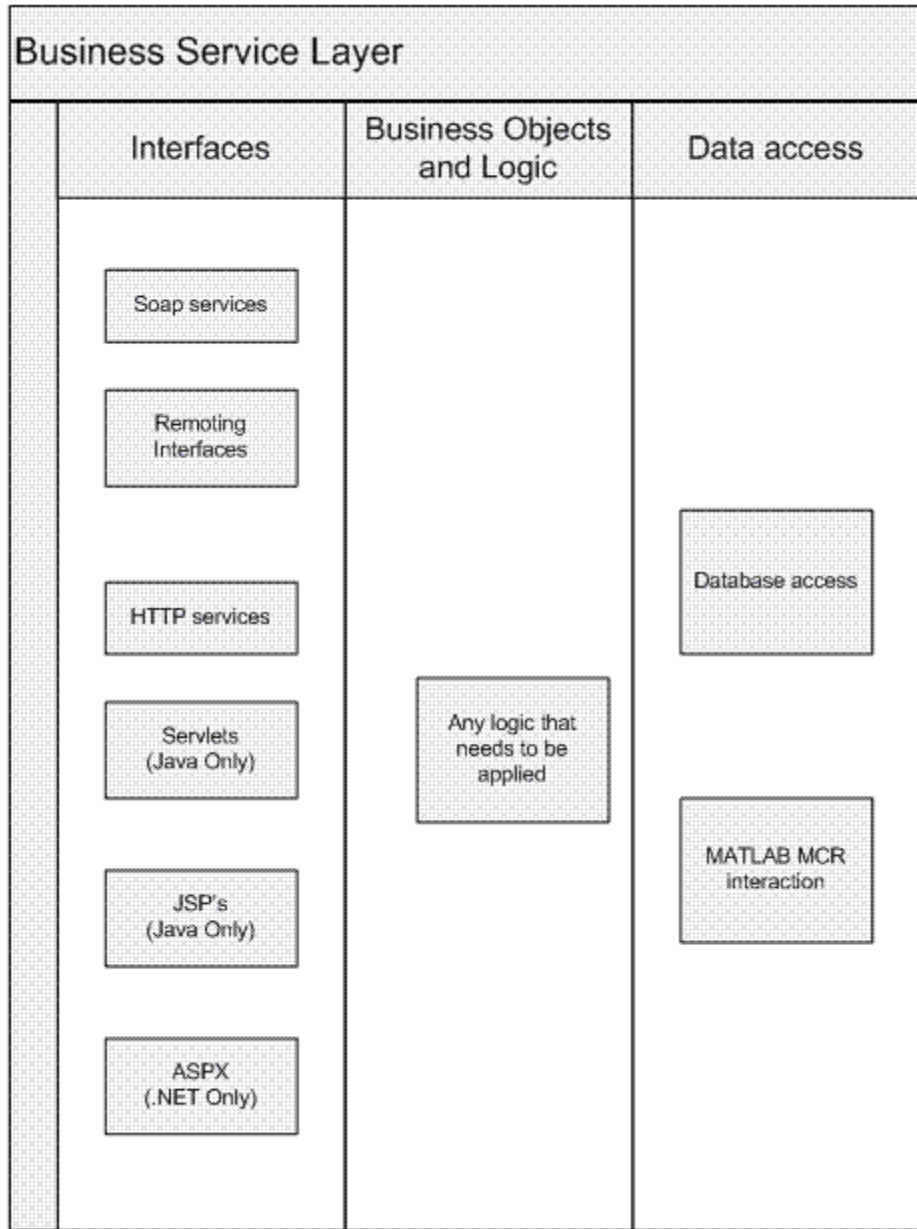
## About the Business Service Layer

Most software contains a *business service layer*: 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 sub-layers:

- 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
- 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 — The data access layer is the link 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**

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

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

## Creating a DAO For Deployment

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

The code in these examples is representative of 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 4-5 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.



**Passing Inputs To a Deployed Application.** Some of the ways to pass inputs to a deployed applications are demonstrated in these examples:

### **Converting an Integers To a MATLAB Data Type In Java**

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

### **Converting an Integers To a MATLAB Data Type In .NET**

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

### **Converting Array Data To a MATLAB Data Type In Java**

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

```
//Create the array of data
String[] friendsArray1 = { "Jordan Robert",
    "Mary Smith",
    "Stacy Flora",
    "Harry Alpert"};
int numberOfArrayElements1 = friendsArray1.length;
int numberOfArrayColumns1 = 1;

//Create the MWCellArray to store the data

MWCellArray cellArray1 = new MWCellArray( numberOfArrayColumns1,
    numberOfArrayElements1);

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

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

### **Converting Array Data To a MATLAB Data Type In .NET**

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

```
String[] array = { "Jordan Robert",
    "Mary Smith",
    "Stacy Flora",
    "Harry Alpert"};
int numberOfArrayElements = array.Length;
int numberOfArrayColumns = 1;

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

### **Converting a List To a MATLAB Data Type In Java**

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

```
//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 MWCCellArray to store the data

MWCCellArray cellArray2 = new MWCCellArray( numberOfListColumns,
    numberOfListElements);
```

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

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

### **Converting a List To a MATLAB Data Type In .NET**

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

```
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;

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

### **Converting a Maps (Name Value Pairs) To a MATLAB Data Type In Java**

In Java 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 as we go.

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

### **Converting a Dictionaries (Name Value Pairs) To a MATLAB Data Type In .NET**

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

```
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 Output From a Deployed Application.** This code resides in the try block for an initialized component (see “Initializing a Component” on page 4-5). Various MATLAB outputs are demonstrated by the following examples:

### Getting Numerics From a Java Component

```

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

```

### Getting Numerics From a .NET Component

```

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

```

### Getting Strings From a Java Component

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

### Getting Strings From a .NET Component

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

### Getting Numeric Arrays From a Java Component

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

### Getting Numeric Arrays From a .NET Component

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

## Getting Character Arrays From a Java Component

```

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

```

## Getting Character Arrays From a .NET Component

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 Java Component

```

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

try
{
    byteOutput = deployment.getByteArray(1);
}

```

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

### Getting Byte Arrays From a .NET Component

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

### Getting Cell Arrays From a Java Component

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

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

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

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

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



## Getting Cell Arrays From a .NET Component

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

```
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 Java Component

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

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

    Map mapOfStruct = new HashMap();

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

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

### Getting Structures From a .NET Component

```
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 Images From a Java Component Using Web Figures

For more information about Web Figures, 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.

---

## Getting Image Data From a Web Figure In a Java Component

This example shows how to get image data from a Web Figure object, as well as how to specify the image type and the orientation of the image.

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

```
Object[] webFigureByteOutput = null;
MWJavaObjectRef webFigureByteReference = null;

try
{
    webFigureByteOutput = deployment.getWebFigure(1);
    webFigureReference = (MWJavaObjectRef)webFigureByteOutput[0];
    WebFigure f = (WebFigure)webFigureByteReference.get();

    Map<String, Object> params = new HashMap();
    params.put( rotation , 30);
    params.put( elevation , 30);
    params.put( width , 500);
    params.put( height , 500);
    params.put( imageFormat , png );

    byte[] webFigureBytes = f.render(params);
}
finally
{
    MWArray.disposeArray(webFigureByteOutput);
    MWArray.disposeArray(webFigureByteReference);
}
```

**Getting Raw Image Bytes From an Image In a Java Component**

```
Object[] byteImageOutput = null;
MWNumericArray numericImageByteArray = null;

try
{
    byteImageOutput =
        deployment.getImageDataOrientation(1, 500, 500, 30, 30, png );
    numericImageByteArray = (MWNumericArray_byteImageOutput[0];
    byte[] imageByteArray = numericImageByteArray.getBytes();
}
finally
{
    MWArray.disposeArray(numericImageByteArray);
}
```

**Getting Raw Image Bytes From an Image In a .NET Component**

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

**Getting a Java Buffered Image In a Java Component**

```
Object[] byteImageOutput = null;
MWNumericArray numericImageByteArray = null;

try
```

```

{
    byteImageOutput =
        deployment.getImageDataOrientation(1, 500, 500, 30, 30, png );
    numericImageByteArray = (MWNumericArray_byteImageOutput[0]);
    byte[] imageByteArray = numericImageByteArray.getBytes();

    BufferedImage bufferedImage =
        ImageIO.read(new ByteArrayInputStream(imageByteArray));
}
catch(IOException io_ex)
{
    io_ex.printStackTrace();
}
finally
{
    MWArray.disposeArray(numericImageByteArray);
}

```

### Getting a .NET Image In a .NET Component

An example of how to get a .NET image back from a byte array follows:

```

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);
byte[] byteArray = (byte[])result.ToVector(MWArrayComponent.Real);

MemoryStream ms = new MemoryStream(myByteArray, 0, myByteArray.Length);
ms.Write(myByteArray, 0, myByteArray.Length);

Image outputImage = Image.FromStream(ms, true);

```

## Hosting a DAO On a Web Server

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

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

## Hosting the DAO With a Servlet

Note that the DAO is initialized in the `init` method of the Servlet. When we 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 directory, if one does not exist, under the directory where your component resides on your Web server. The DAO must reside in this directory, 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 could 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 (Web Figures)**

This example is very similar to what can be found in “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));
}
```

### **Web Figure 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 do similar functionality to what Web Figures (see “Deploying a Java Component Over the Web”) 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.getImageByteArrayFromMWNumericArrayWithOrientation(
        height, width, elevation,
        rotation, imageFormat);
response.getOutputStream().write(rawImageBytes);
```

- 3** Inside the staging directory you created at the start of this procedure, create a WEB-INF directory.
- 4** Inside the WEB-INF directory, create two additional directories:
  - classes
  - lib
- 5** Place all of the class files (including the DAO created in “Creating a DAO For Deployment” on page 4-5) into the class directory within of the appropriate package directories 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 directories you created into a WAR (Web archive) and place it in your Web server's component directory.

---

**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 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 (The Web Service Definition Language — the definition of your service) and generate Java from that. Afterwards, 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 POJO's (Plain Old Java Objects) we 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.** To deploy the Web service, follow these steps:

- 1 Create a staging directory, if one does not exist, and copy the Examples DAO class created in “Creating a DAO For Deployment” on page 4-5 and the Web service class created in into it.
- 2 Create a lib directory 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 directory (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 do similar functionality to what Web Figures 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);
```

**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 directory.

### 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][];
    for (int i = 0;
        i < (int)doubleMatrix.GetUpperBound(0) + 1; i++)
    {
        double[] subArray = new double[arraySize];
        for (int j = 0;
            j < (int)doubleMatrix.GetUpperBound(1) + 1; 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  
(p. 5-2)

Examples of tasks to perform while  
at the front end layer

## Working With the Front End Layer

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

### In this section...

“Creating a Web Figure On a JSP Page” on page 5-5

“Working With Static Images” on page 5-8

“Displaying Complex Data Types Including Arrays and Matrices” on page 5-12

“Using Web Services” on page 5-19

---

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

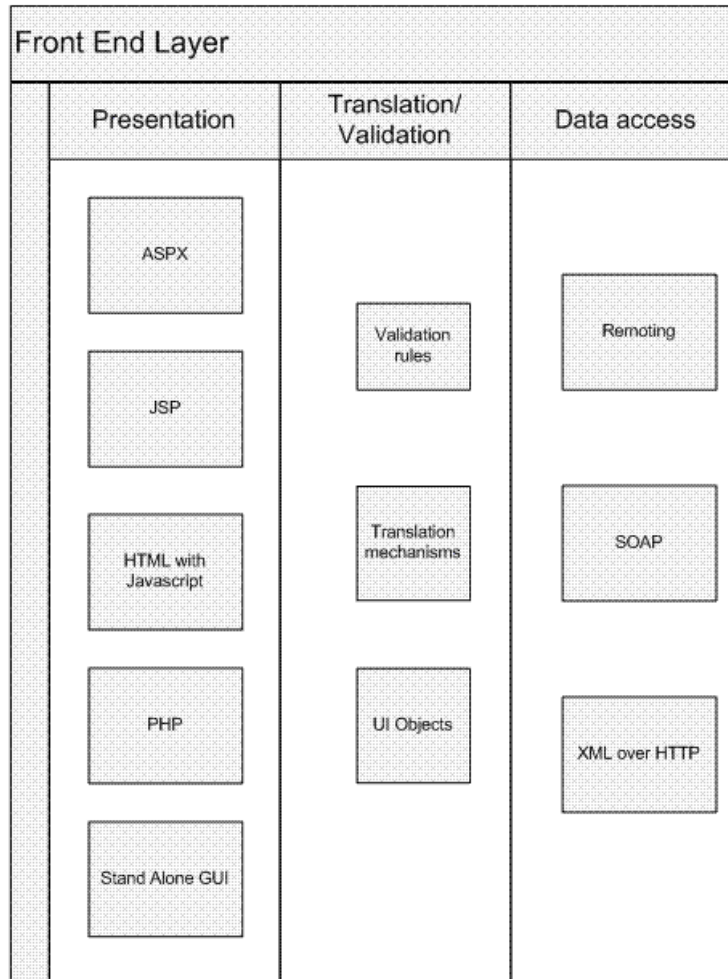
---

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 will have 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 the changing user visible code.

A typical front end layer contains the following sub-layers:

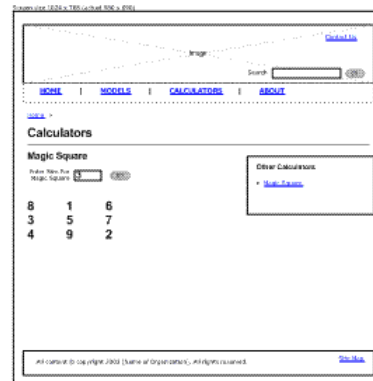
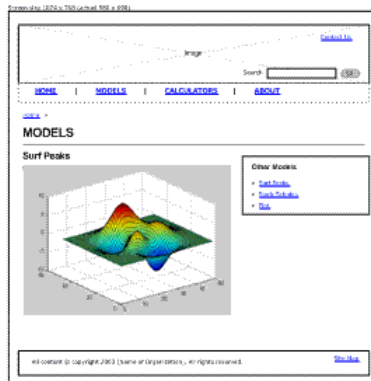


### Elements Of the Front End Layer

- **Data Access** — The data access sub-layer is used to pull 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 sub-layer are:

- Remoting interfaces
- SOAP services
- XML over HTTP protocol
- Translation/Validation — Data is passed from the data access sub-layer to the translation sub-layer 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 — The presentation 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.

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 two templates in Surf Peaks and Magic Square Application Integration on page 5-4 show how applications built with MATLAB products can be

integrated into a larger application. In each case, there is a small area in the interface where our 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 sub frame, or to use embedded Web Figures.

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 off 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 which follow, these concepts will be simplified and focus on how the communication occurs within the middle layer, and how some typical data translations are performed.

## Creating a Web Figure On a JSP Page

There are several ways to utilize Web Figures on a Web page with Java:

- Return a special HTML string from the Servlet which embeds the Web Figure into a page.
- Using the custom Web figure tag library directly from the JSP. have the Servlet bind the Web Figure 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 Web figures functionality and the correct call backs directly into the front end page.

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

### Using an HTML String

This example is very similar to what can be found in "Deploying a Java™ Component Over the Web", but this example also uses our DAO. The middle

tier code featured in “Hosting the DAO With a Servlet” on page 4-20 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 Web Figure object directly from a JSP page, reference the webfigures tag library. This creates a Web Figure object with your object’s parameters. The UserPlot is the name of the Web Object that was placed in the cache by the middle tier.

---

**Note** The middle tier and the JSP must be hosted on the same server.

---

```
<%@ taglib prefix="wf" uri="/WEB-INF/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 side 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.

---

**Note** 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, the front end can be coded 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:

```
WebFigures webfigures =
    new WebFigures("WebFigures", getServletContext());

String embeddedString;
try
{
    //This generates a string that can be sent to the
    // response that represents the web figure.
    embeddedString =
        webfigures.getHtmlEmbedString( userPlot,
```

```
        "UserPlot",
        "session",
        null,
        null,
        null);
    }
    catch(MWException mwe)
    {
        throw new Exception();
    }

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

## Working With Static Images

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

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

```

```



## 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 4-34. 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 Web Figure On a JSP Page” on page 5-5 is a good solution for most component models, sometimes a lightweight solution is needed that may be customized 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 the DAO With a Servlet” on page 4-20 and . 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];
        }

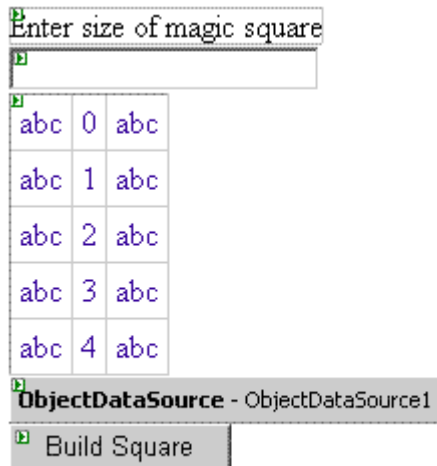
        //Add the row to the table
        table.Rows.Add(row);
    }
    return table;
}
```

### **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 Visual Studio’s wizards and a typical Web page application. Perform the following steps to set up the data source,

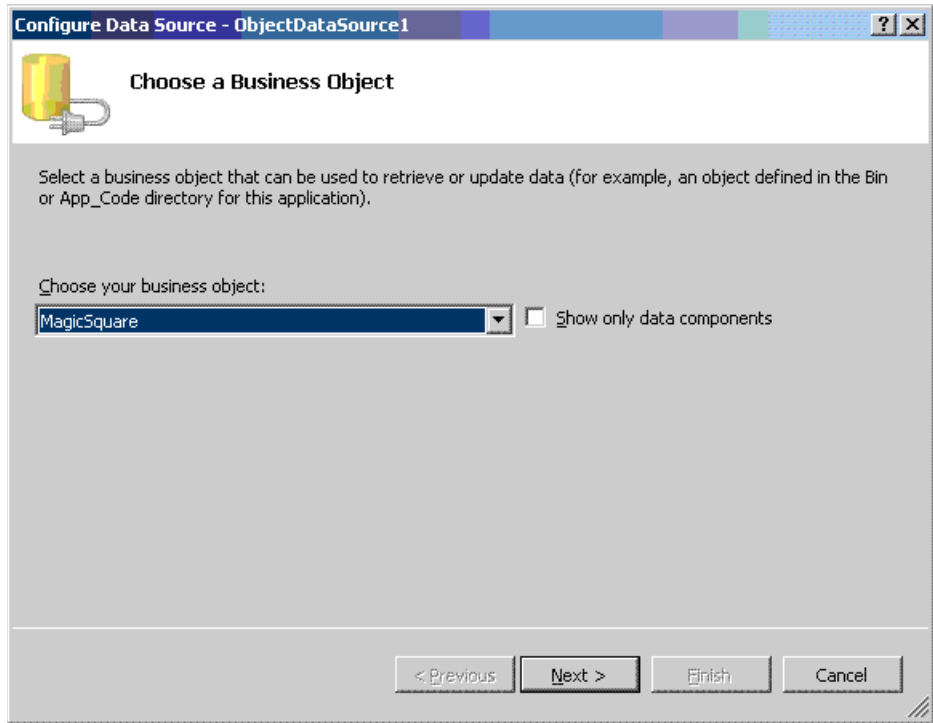
bind it to a load method, and bind the method's input parameter to the text box.

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

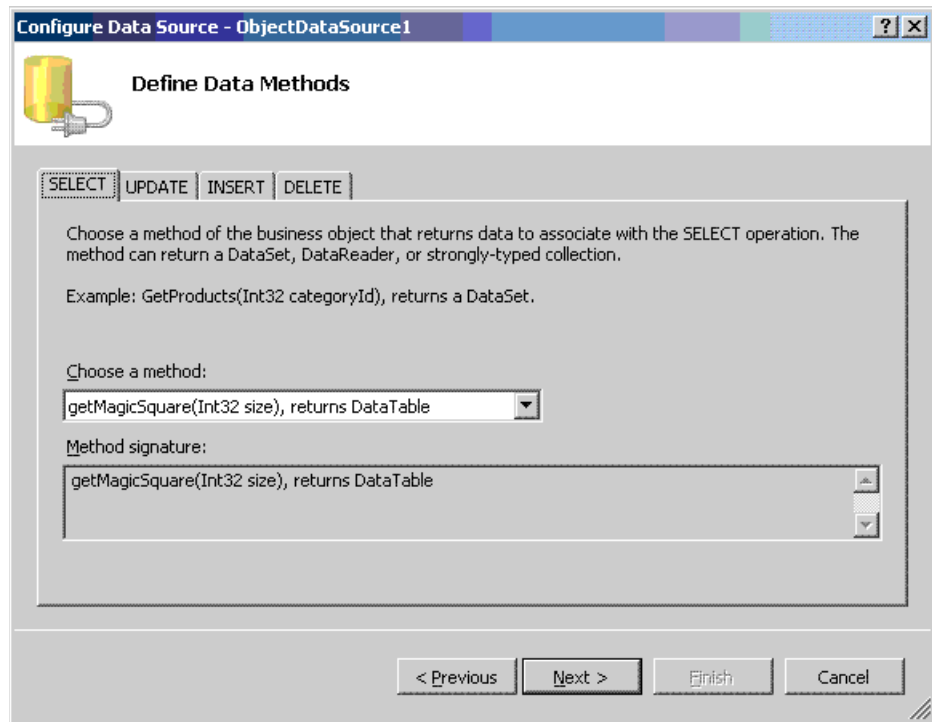


### Input Text Box and Grid Component

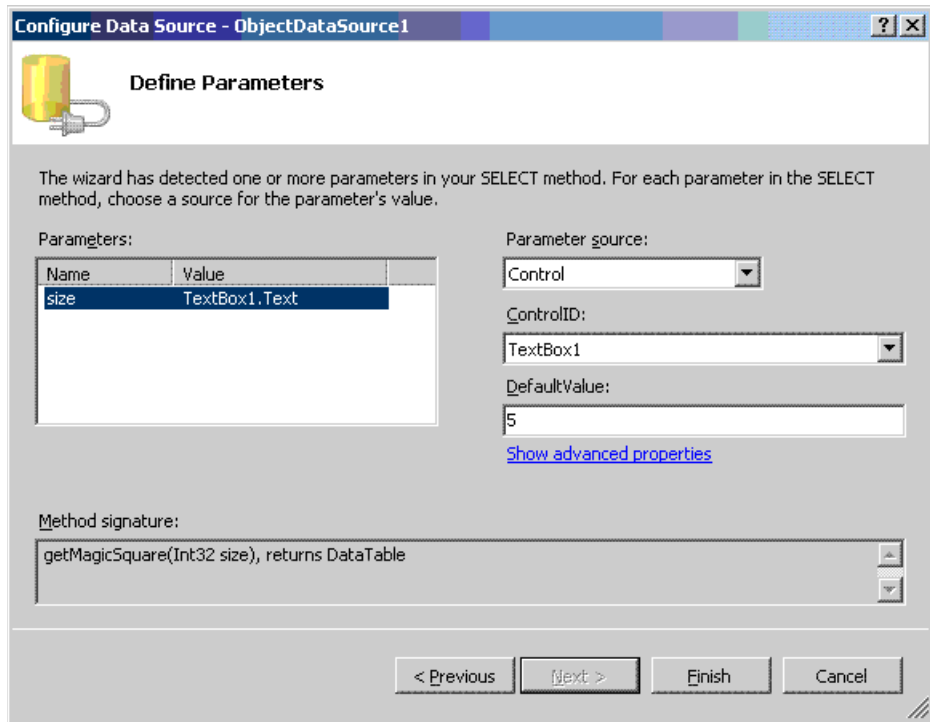
- 1 Start Visual Studio and configure the data source with the wizard.
- 2 Choose the business class that contains your methods for the object:





**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

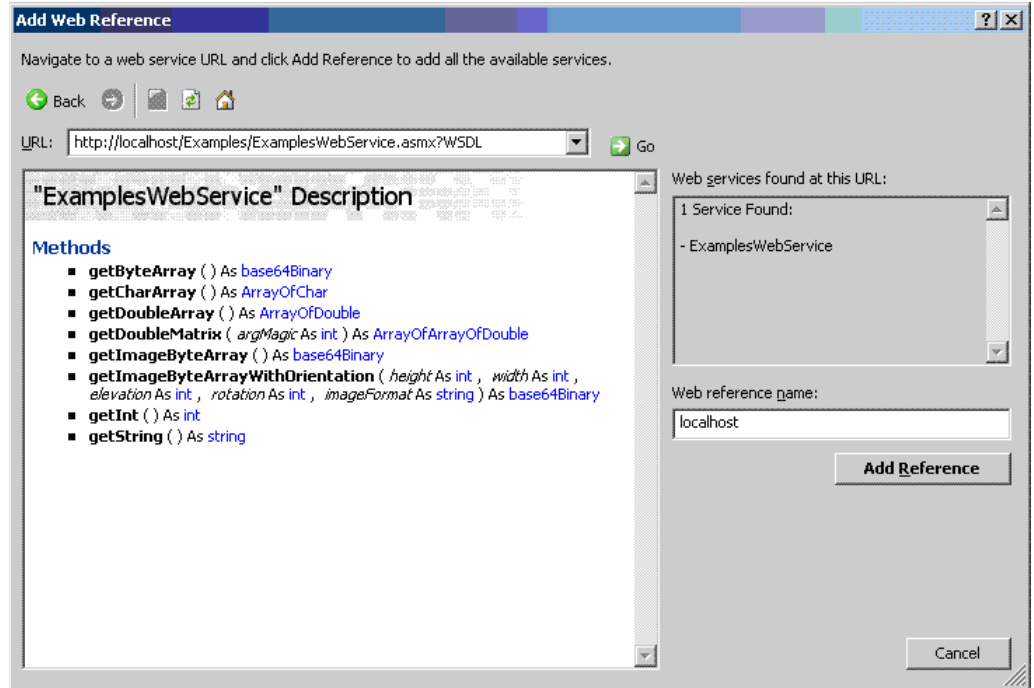
17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9
Build Square				

The **Build Square** button reselects the grid data.

## Using Web Services

In Chapter 4, “Business Service Developer Tasks”, we 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 on 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 should look something like 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();
```

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

```

## 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 non language 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 (somewhere the include command can see it)

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.

# Internal Analyst Tasks

---


Working With the Content (p. 6-2)

The role of the Internal Analyst and common workflows which they perform

Examples of Internal Analyst Tasks (p. 6-3)

Examples of common Internal Analyst tasks

## Working With the Content

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

Internal Analyst 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.



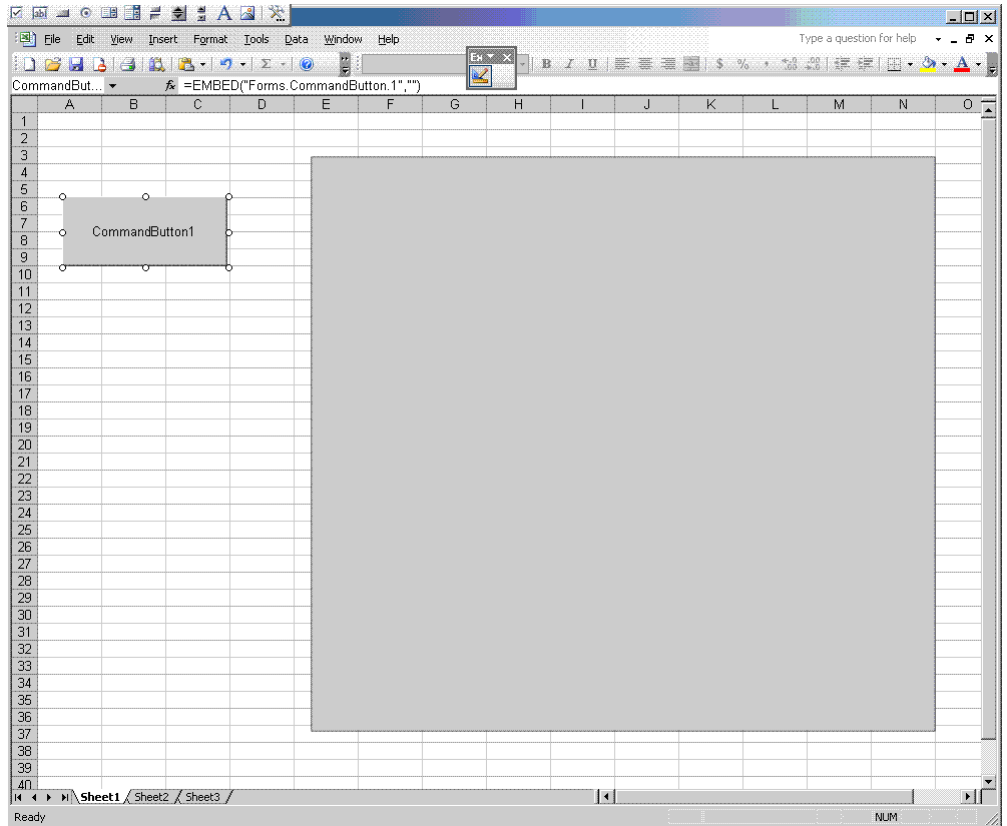
## Examples of Internal Analyst Tasks

The example in this section consumes the .NET Web service created in “Deploying the Web Service” on page 4-37. Using this type of application, you can use the same Web service to display many different front ends. You can use data stored in Microsoft Excel and pass it to a Web service to generate dynamic data driven images.

### **A Microsoft Excel Web Service Client Standalone .NET Application**

To construct a Microsoft Excel interface to the Web service, do the following:

- 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. This button will be used 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...**

---

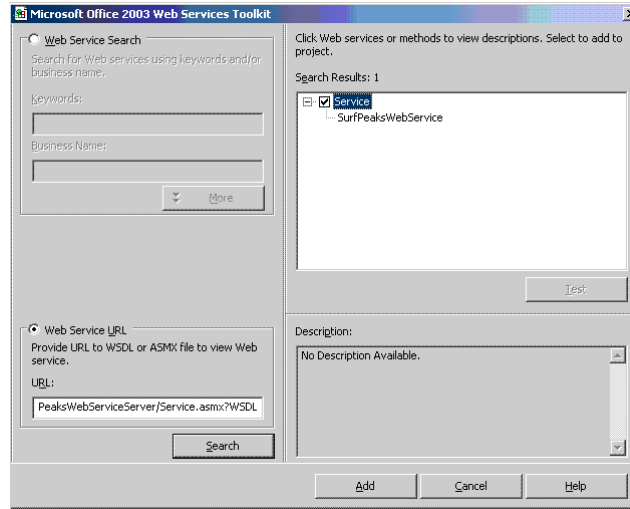
**Note** The Web Service References... option is only available if you install the Microsoft Office Web Service Toolkit.

---

**8** In **Web Service URL**, type the WSDL that we referenced in “Using Web Services” on page 5-19:

<http://localhost:3465/SurfPeaksWebServiceServer/Service.asmx?WSDL>

- 9 Click **Search** to query the Web service. The result will look like 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
```

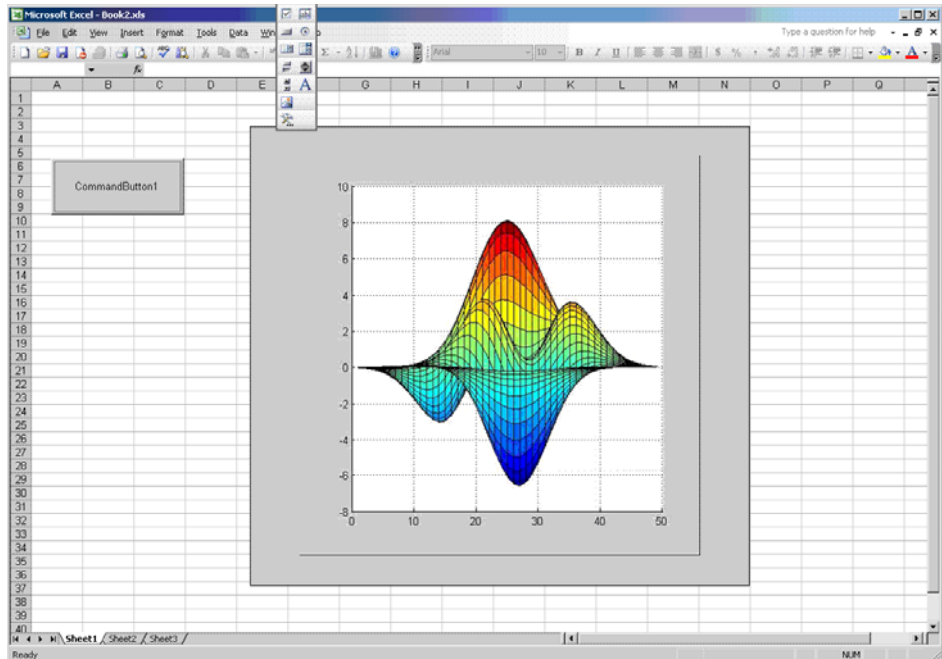
```
'Loads the saved picture into the image  
Sheet1.Image1.Picture = LoadPicture("c:\temp1.png")
```

- 12** Click the command button and the following image appears in the graphics pane of your worksheet:

---

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

---



# Sources for More Information

---

[Other Examples \(p. A-2\)](#)

[Links more for information about the tools and concepts in this guide](#)

## 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 Shoals**

MATLAB Central Black Shoals Web Demo For Java

#### **Web Figures**

MATLAB Builder JA Web Figures Varargs Demo

### **MATLAB® Builder™ NE**

Other examples using MATLAB Builder NE include:

#### **Black Shoals**

MATLAB Central Black Shoals Web Demo For .NET

## **B**

business service layer 4-2

## **D**

DAO 4-5

  Hosting

    Using a .NET Web Service 4-34

    Using a Java Web Service 4-26

  Hosting On a Web Server 4-20

  Initializing

    For a Servlet 4-21

  Initializing the

    with ASPX 4-31

  Interacting with 4-32 4-34

    in Java 4-27

  Using an HTML String 5-5

## **F**

front end layer 5-2

## **N**

NUSOAP

  Download location 5-22

NUSOAP PHP class add-in 1-5

## **S**

SOAP 4-26

SOAP Web services

  As part of data access in front-end layer 5-4

  As part of interfaces in business service  
  layer 4-2

  Using Embedded HTML with 5-7

  Using To embed an image on a PHP  
  page 5-21

  Using Web references with 5-19

SOAP Web Services 1-2

## **W**

Web Figures 2-2 4-5

  and MATLAB figures 3-3

  Deploying a Java component 4-24

  Example

    using a DAO 4-24

  Getting image data from a Web figure  
  Java component 4-17

  Getting images from a Java component  
  using 4-16

  JSP page 5-5

  Other examples of A-2

  Returning a figure as data 3-4

  Returning data from a Web Figure  
  Window 3-3

  Use in AJAX component 5-5

  Using directly from JSP

    Using tag library 5-6