



### Domino development with servlets

by  
Anthony  
Patton

**Level:** Advanced  
**Works with:** Domino 5.0  
**Updated:** 02/01/2001

Domino R5 support for Java servlets increases the choices you have when developing applications. You can use servlets instead of Domino agents to access Domino objects and accomplish server-side processing, for example.

This article provides basic information about servlets, including the differences between agents and servlets, setting up your server to work with servlets, and basic servlet structure. It then explores three servlet examples in detail.

This article assumes familiarity with JavaScript and an understanding of designing Domino applications.

### Some servlet basics

A servlet is a Java program that runs on a server and offers functionality similar to CGI (Common Gateway Interface) applications and Domino agents. The main advantage of a servlet is that it is loaded into memory once, as opposed to Domino agents and CGI programs that are loaded for each call. All calls to a servlet, after its initial load, use the same instance.

While Domino agents and servlets offer similar functionality, they are very different in terms of implementation. An agent resides in a Domino database while servlets reside on the file system. This brings two issues immediately to mind: distribution and security.

A Domino agent can take advantage of all aspects of its database container. Therefore, replication can be used for distribution to other Domino servers. Likewise, the Domino security model determines agent access.

Conversely, servlets live on the file system so server, file, and directory security controls access. By default servlets run with the security rights of the server, but they can run using a specific person's Internet name and password. Finally, unless you work with directory replication on Windows NT you must copy or install a servlet's files on all other systems desired.

### About the JSDK

The standard Domino Designer development environment does not support the development of servlets, so you must use a third-party IDE or the SUN command-line JDK to develop servlet code. (See the *Iris Today* article [Creating servlets for Domino with VisualAge for Java](#) for a discussion of servlet development with IBM's VisualAge for Java.)

The classes required to develop a servlet are freely available from [Sun Microsystems](#) in the Java Servlet Development Kit (JSDK). In addition, the JSDK is a standard part of a Domino install. The jsdk.jar file is located in your root Domino installation directory. For example, my Domino server is installed in the directory r5server on my D drive, so the path to my copy of the file is:

D:\r5server\jsdk.jar

**Note:** Jar files are the compression standard used for Java files. They are similar to WinZip, PKZip, or Windows CAB files .

## Server setup

Domino R5 comes with its own servlet manager, but you can also use a third-party servlet engine such as IBM WebSphere (currently, WebSphere is the only supported engine). The Server document in the Domino Directory contains a section for servlet settings. These settings are located in the Java Servlets section of the Domino Web Engine tab on the Internet Protocols tab. Here are the servlet settings on my test server, for example:

The screenshot shows the Domino Server document settings for the Internet Protocols tab. The 'Domino Web Engine' sub-tab is selected. The 'Java Servlets' section is expanded, showing the following settings:

Section	Setting	Value
HTTP Sessions	Session authentication:	Disabled
	Idle session timeout:	30 minutes
	Maximum active sessions:	1000
	Generating References to this Server	
Does this server use IIS?	Does this server use IIS?	No
	Protocol:	Http
	Host name:	
	Port number:	80
Memory Caches	Maximum cached commands:	128
	Maximum cached designs:	128
	Maximum cached users:	64
	Cached user expiration interval:	120 seconds
Java Servlets	Java servlet support:	Domino Servlet Manager
	Servlet URL path:	/servlet
	Class path:	domino/servlet
	Servlet file extensions:	
Session state tracking	Session state tracking:	Enabled
	Idle session timeout:	30 minutes
	Maximum active sessions:	1000
	Session persistence:	Disabled
POST Data	Maximum POST data (in kilobytes):	0
	File compression on upload:	Disabled

Let's take a closer look at the options available:

Option	Description
Java servlet support	This enables/disables servlet support. The possible values are None, Domino Servlet Manager, and Third-Party Servlet Support.
Servlet URL path	This is used to indicate the URL string that can be used to access servlets. This is the directory in which servlet related files must be located to properly run.
Class path	This is the location of any servlets to load and run. This is a local directory. The Domino Servlet Manager locates any servlets that are listed in the classpath directory.
Servlet file extensions	This is a list of URL file extensions that tell Domino that it is a servlet. A comma separates multiple extensions.
Session state tracking	This setting signals (enabled or disabled) whether the servlet manager should terminate idle sessions that are initiated by Java calls from servlets. The idle session time limit is set in the next field.
Idle session timeout	This specifies the amount of time (in minutes) a connection can be idle before it is terminated. This is used only if Session state tracking is enabled. The default is 30.
Maximum active sessions	This is the number of active servlet sessions allowed on the server at any one

	time. The default is 1000.
Session persistence	This signals (enabled or disabled) whether or not session information is saved to disk.

Domino servlet support includes two pieces: the JVM (Java Virtual Machine) and the Servlet Manager. When servlets are enabled, the JVM loads moments before the HTTP server starts. Consequently, if the Domino Servlet Manager is used, it is loaded after the JVM is loaded. If you are using a third-party servlet manager, only the JVM is loaded.

## The servlet properties file

The Domino servlet engine includes support for a configuration file that specifies standard parameters when it is loaded. It is a text file named `servlets.properties`, and it is located in the data directory of your Domino installation, for example:

`D:\r5server\data\servlets.properties`

If you can't find the file, you can easily create it with any text editor, such as Windows Notepad.

The servlet properties file lets you specify an alias, initialization arguments, URL extension mapping, and the loading of servlets upon Web server startup. You can insert comments as well.

### Servlet aliases

The alias directive has this syntax:

```
servlet.<alias-name>.code=<class-name>
```

### Initialization arguments

You can specify initial data for a servlet in the properties file. The servlet can access the data by using the method `ServletConfig.getInitParameter`. The initialization directive has this syntax:

```
servlet.<alias or class name>.initArgs=<name1=value1>,<name2=value2>
```

### URL extension mapping

The URL extension-mapping directive has this syntax:

```
servlet.<alias or class name>.extension=<extension> <extension> ...
```

### Load on startup

The startup directive has this syntax:

```
servlets.startup=<alias or class> <alias or class> ...
```

Here is the content of a `servlets.properties` file that assigns a name (test) to a servlet named `HelloWorld.class` (notice the class extension is optional). Initial arguments are assigned to it, and it is loaded upon startup.

```
# Example servlets.properties file
servlet.Test.code = HelloWorld
servlet.Test.initArgs = 1, 2, 3
servlets.startup = Test
```

The great aspect of loading a servlet on Web server startup is that it is available in memory from that point forward. Keep in mind, however, that if any changes are made to a servlet's code, the Domino HTTP server must be shut down and restarted for the changes to be recognized. The restart

command can be used:

**Tell HTTP restart**

### Common mistakes

Here is a brief list of common mistakes encountered when working with a servlet configuration file:

- Placing the file in the wrong directory; it must be located in your Domino data directory.
- Using the wrong file extension; many editors like Notepad will append the .TXT file extension.
- Using an incorrect file name; the file name must be `servlets.properties`.
- Ignoring case-sensitivity; the file name is case-sensitive, as are servlet names and assignment statements.

## The Domino configuration file

Once servlets are enabled on a Domino server, several servlet-related lines will appear in the Domino configuration file (`domino.cnf`) located in the data directory. Here are those lines:

**# Java Servlet Settings**

```
ServerInit servlet:ServletInit d:\r5server\Data
Service /servlet/* servlet:ServletService*
ServerTerm servlet:ServletTerm
```

## The servlet life cycle

As previously stated, a servlet is loaded into memory once and only once. When it is loaded, the `init` method is executed. The `service` method is executed every time a client requests a servlet, and the `doPost` or `doGet` method is called depending on the client request.

Servlets can be loaded via two methods: the Servlet Manager loads at HTTP startup as specified in the `servlets.properties` file or at the initial request from a client. A servlet is disposed (cleared from memory) when the HTTP task is shut down.

You should use the Servlet Manager as much as possible via the `servlets.properties` file. This places the load on the server at HTTP startup, and client requests encounter no delay. The response time will be somewhat the same the first and last time a servlet is requested.

## Servlet structure

HTTP requests can be of two types: `get` and `post`. The `get` method appends information to the end of the URL. It is available in the `query_string` environment variable. `Post` sends information in the form of `name/value` pairs.

When developing servlet code, there are two packages from the JSDK that are used to work with `post` or `get` methods, `javax.servlet` and `javax.servlet.http`. These packages are imported for use in your Java servlet.

There are three methods of a servlet that are important to understand:

- `service`  
This is called each time a server requests a servlet. Usually, this method is not used for HTTP servlets; it is more applicable to generic servlets.
- `doPost`  
All `post` requests issued from a Web server to the servlet invoke this method. Its format is:

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
```

- **doGet**

All get requests issued from a Web server to the servlet invoke this method. Its format is:

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
```

The two methods to become most acquainted with are `doPost` and `doGet`. Each method accepts two object parameters, `HttpServletRequest` and `HttpServletResponse`.

### HttpServletRequest

The `HttpServletRequest` object contains all information from the requesting client. This object can be used to access request headers, CGI variables, cookies, and form data, and to conduct session tracking. Actually, it parses incoming form data and stores it in servlet parameters.

Here is a partial list of `HttpServletRequest` class methods:

Method	Description
<code>getContentLength</code>	Returns the length of content in bytes.
<code>getContentType</code>	Returns the MIME type for content.
<code>getCookies</code>	Returns an array of cookie objects.
<code>getMethod</code>	Returns the HTTP method utilized.
<code>getParameter(String name)</code>	Returns the value of a specified parameter. The servlet engine places all HTML form values into parameters. Use this method for single-value pairs.
<code>getParameterValue(String name)</code>	Returns all values for a specified parameter. Use this when a parameter has more than one possible value.
<code>getQueryString</code>	Returns the query string from the requestor URL.
<code>getRemoteUser</code>	Returns the name (if any) of the requestor.

### HttpServletResponse

The `HttpServletResponse` object contains all communication to a client or requester from the servlet. Here is a partial list of methods:

Method	Description
<code>getWriter</code>	Returns a <code>PrintWriter</code> object for the purpose of sending response data to the requestor.
<code>sendRedirect(String url)</code>	Redirects the response to a specified location.
<code>setContentType(String type)</code>	Sets the type of content that is sent to the requestor.
<code>setStatus(int code)</code>	Sets the HTTP status code.

## Creating and running a servlet

Now let's take a look at a simple servlet. The first four lines import necessary information. The first imports Java classes for sending output to the requesting client, and the second imports the necessary Java classes for handling input/output errors. The third and fourth lines import Java Servlet classes.

```
import java.io.PrintWriter;  
import java.io.IOException;  
import javax.servlet.*;  
import javax.servlet.http.*;
```

Next is the servlet class declaration, followed by the doGet method.

```
public class Example_1 extends HttpServlet  
{  
    public void doGet(HttpServletRequest req,HttpServletResponse res)  
        throws ServletException,IOException  
    {
```

We are working with browser clients, so next we set the content-type of the output to HTML. Then we initiate the PrintWriter object, which will be used to send the content to the browser.

```
        res.setContentType("text/html");  
        PrintWriter toBrowser = res.getWriter();
```

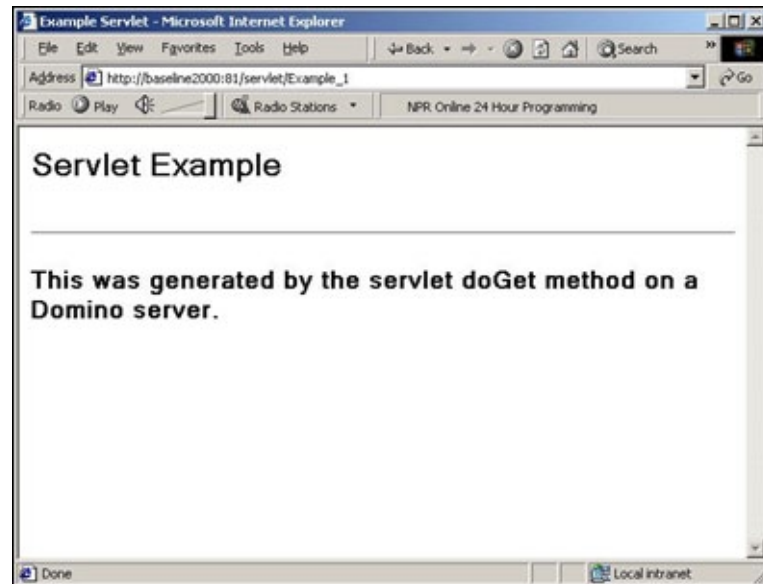
The next section sends the HTML to the browser.

```
        toBrowser.println("<HTML>");  
        toBrowser.println("<HEAD>");  
        toBrowser.println("<TITLE>Example Servlet</TITLE>");  
        toBrowser.println("</HEAD>");  
        toBrowser.println("<BODY>");  
        toBrowser.println("<H1>Servlet Example</H1>");  
        toBrowser.println("<BR><HR><BR>");  
        toBrowser.print("<H2>This was generated by the servlet");  
        toBrowser.println("<doGet method on a Domino server.</H2>");  
        toBrowser.println("</BODY></HTML>");  
    }
```

Finally, the doPost method completes the process.

```
        public void doPost(HttpServletRequest req, HttpServletResponse  
        res)  
            throws ServletException, IOException  
        {  
            res.setContentType("text/html");  
            PrintWriter toBrowser = res.getWriter();  
            toBrowser.println("<HTML>");  
            toBrowser.println("<HEAD>");  
            toBrowser.print("<TITLE>Example - Servlet</TITLE></HEAD>");  
            toBrowser.println("</HEAD>");  
            toBrowser.println("<BODY>");  
            toBrowser.println("<H1>Servlet Example</H1>");  
            toBrowser.println("<BR><HR><BR>");  
            toBrowser.print("<H2>This was generated by the servlet");  
            toBrowser.println("<doPost method on a Domino server.</H2>");  
            toBrowser.println("</BODY></HTML>");  
        }  
    }
```

Here are the results of calling the servlet from Internet Explorer:



And here's what happens on the Domino server when the servlet is called. Notice that here, the Java Servlet Manager loaded when the request was sent to the server by the client.

```

Lotus Domino Server: Baseline1/Baseline
Lotus Domino r Server, Release 5.0.5 , September 22, 2000
Copyright c 1985-2000, Lotus Development Corporation, All Rights Reserved
11/17/2000 04:43:34 PM Server started on physical node BASELINE2000
11/17/2000 04:43:35 PM An Adminp request has been submitted to update port
information in the server document
11/17/2000 04:43:36 PM JVM: Java Virtual Machine initialized.
11/17/2000 04:43:36 PM Java Servlet Manager initialized
11/17/2000 04:43:37 PM Domino Off-Line Services HTTP extension (Release 1.0
loaded.
11/17/2000 04:43:38 PM HTTP Web Server started
11/17/2000 04:43:40 PM DIIOP Server started on baseline2000
11/17/2000 04:43:45 PM Database Server started
11/17/2000 04:44:07 PM Addin: Agent printing: Example_1: init
>

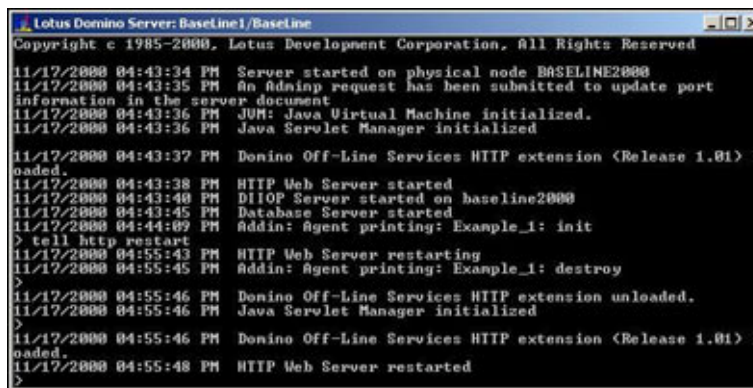
```

The "Addin: Agent printing: Example\_1: init" in the last line signals the servlet was loaded into memory on the server. Every subsequent call will use this instance. You can test this by making repeated requests to the servlet (you may want to shut down and restart your browser).

### Changing a servlet

An important point is the procedure for making changes to a servlet. If you make changes to your servlet code and recompile it, the Domino server must reload the servlet. This is not done automatically, but it can be forced by restarting the HTTP task. The following screen shows restarting the HTTP task on my server. Notice that the servlet is destroyed; it will be reloaded once a client requests it.





```

Lotus Domino Server: Baseline1/Baseline
Copyright c 1985-2000, Lotus Development Corporation, All Rights Reserved
11/17/2000 04:43:34 PM Server started on physical node BASELINE2000
11/17/2000 04:43:35 PM An Adminp request has been submitted to update port
information in the server document
11/17/2000 04:43:36 PM JVM: Java Virtual Machine initialized.
11/17/2000 04:43:36 PM Java Servlet Manager initialized
11/17/2000 04:43:37 PM Domino Off-Line Services HTTP extension (Release 1.01) l
oaded.
11/17/2000 04:43:38 PM HTTP Web Server started
11/17/2000 04:43:40 PM DIIOP Server started on baseline2000
11/17/2000 04:43:45 PM Database Server started
11/17/2000 04:44:09 PM Addin: Agent printing: Example_1: init
> tell http restart
11/17/2000 04:55:43 PM HTTP Web Server restarting
11/17/2000 04:55:45 PM Addin: Agent printing: Example_1: destroy
>
11/17/2000 04:55:46 PM Domino Off-Line Services HTTP extension unloaded.
11/17/2000 04:55:46 PM Java Servlet Manager initialized
>
11/17/2000 04:55:46 PM Domino Off-Line Services HTTP extension (Release 1.01) l
oaded.
11/17/2000 04:55:48 PM HTTP Web Server restarted
>

```

### Loading the servlet on server startup

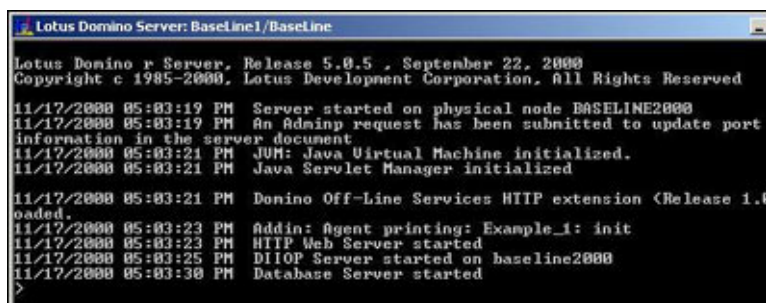
Another situation is the loading of the servlet at server startup. You can configure your Domino server via the `servlets.properties` file to load your servlet at startup. For example, if its name is `NotesNetTest`, here are the contents of the properties file:

```

servlet.NotesNetTest.code = Example_1
servlets.startup = NotesNetTest

```

The following screen shows the Domino server console on startup. Notice the servlet loads just before the HTTP server task.

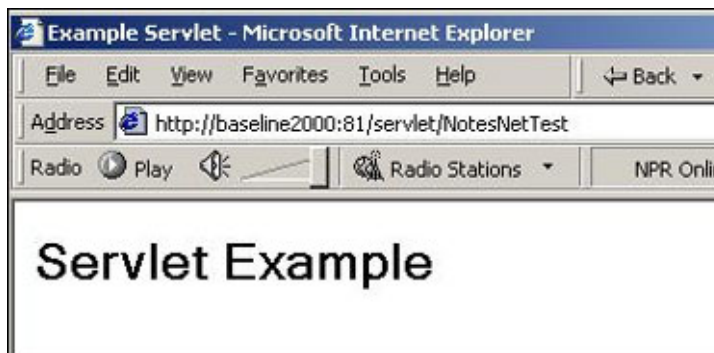


```

Lotus Domino Server: Baseline1/Baseline
Lotus Domino r Server, Release 5.0.5 , September 22, 2000
Copyright c 1985-2000, Lotus Development Corporation, All Rights Reserved
11/17/2000 05:03:19 PM Server started on physical node BASELINE2000
11/17/2000 05:03:19 PM An Adminp request has been submitted to update port
information in the server document
11/17/2000 05:03:21 PM JVM: Java Virtual Machine initialized.
11/17/2000 05:03:21 PM Java Servlet Manager initialized
11/17/2000 05:03:21 PM Domino Off-Line Services HTTP extension (Release 1.0
loaded.
11/17/2000 05:03:23 PM Addin: Agent printing: Example_1: init
11/17/2000 05:03:23 PM HTTP Web Server started
11/17/2000 05:03:25 PM DIIOP Server started on baseline2000
11/17/2000 05:03:30 PM Database Server started
>

```

We can now call the servlet using the name `NotesNetTest`, as the following screen illustrates.



### Working with Domino objects

Now let's turn our attention to tighter integration of servlets and Domino. The next example uses Domino objects in a servlet. It accesses entries in the Domino Directory.

Initially, the code resembles the previous example, although here, the import `lotus.domino.*` line imports the necessary Domino Java classes.



```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;
public class Example_2 extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException{
        res.setContentType("text/html");
        PrintWriter toBrowser = res.getWriter();
        toBrowser.println("<HTML>");
        toBrowser.println("<HEAD>");
        toBrowser.println("<TITLE>Example 2</TITLE>");
        toBrowser.println("</HEAD>");
        toBrowser.println("<BODY>");
        toBrowser.println("<H1>Example 2</H1>");
    }
}

```

Calls to Domino objects must be contained within a try/catch block, so the try { line is next. Within this block, we create a NotesThread object for accessing the Domino objects. Then we create a new Session object. There are no parameters for the Session object, so the server's access is used, but we could have specified a user's Internet name and password instead. Next, we access the Domino Directory (names.nsf) and the People view, and then we retrieve the first document from the view.

```

try {
    NotesThread.sinitThread();
    Session s = NotesFactory.createSession();
    Database db = s.getDatabase("", "names.nsf");
    View vw = db.getView("People");
    Document doc = vw.getFirstDocument();
    toBrowser.println(db.getTitle());
    toBrowser.println("<TABLE>");
    while (doc != null)
    {
        toBrowser.println("<TR><TD>");
        toBrowser.println(doc.getItemValueString("LastName"));
        toBrowser.println("</TD></TR>");
        doc = vw.getNextDocument(doc);
    }
    toBrowser.println("</TABLE>");
    vw.recycle();
    db.recycle();
    s.recycle();
}
catch (NotesException n) {
}

```

The toBrowser lines send the database title to the requesting client (the browser). With the while (vw!=null) line, we begin to loop through all the documents in the view, displaying each document's Last Name value in the browser and then retrieving the next document in the view. After the last document, we release (recycle) the memory used by the Domino objects to the system.

Finally, we display any Notes errors that were encountered and terminate the NotesThread object in the finally block.

```

        System.out.println("Exception ID: " + n.id);
        System.out.println("Exception description: " + n.text);
    }
    finally
    {
        NotesThread.stermThread(); }
}

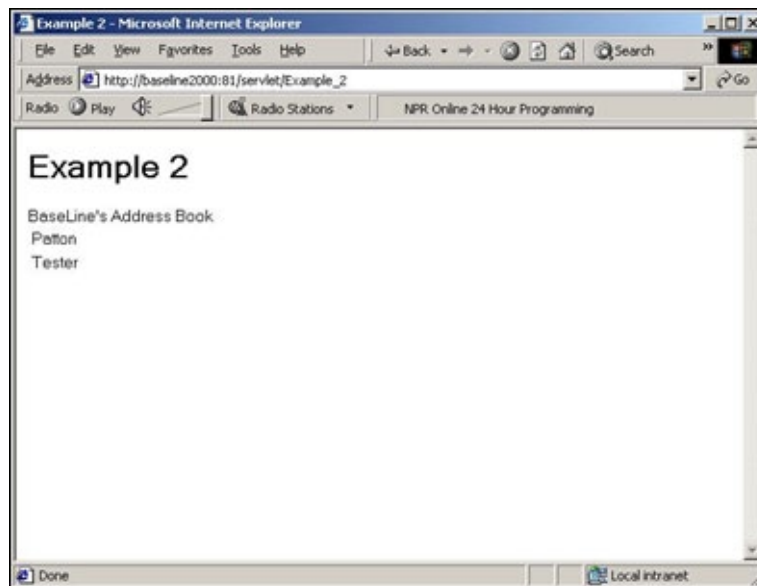
```

```

        toBrowser.println("</BODY></HTML>");
    }
}

```

The following screen shows the results of accessing this servlet on a test server:



## Combining with Domino forms

Now that we've seen how to build a servlet and incorporate Domino objects, let's turn our attention to the combination of a servlet and Domino forms. We'll examine the code first, followed by the setup of a form.

The following code comprises a servlet that handles values submitted from a form. The values are retrieved from the form and used to populate a new entry in a Domino database. The code begins with the "standard" lines. Then we retrieve the values of the FName, LName, Email, and Phone fields from the form.

```

import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;
import java.io.PrintWriter;
import java.io.IOException;
public class Example_3 extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter pw = res.getWriter();
        pw.println("<html>");
        pw.println("<head>");
        pw.println("<title>Example 3</title>");
        pw.println("</head>");
        pw.println("<body>");
        pw.println("<h1>Thank you!</h1>");
        String fname = req.getParameter("FName");
        String lname = req.getParameter("LName");
        String email = req.getParameter("Email");
        String phone = req.getParameter("Phone");
    }
}

```

Next, as we did in the previous servlet, we create a NotesThread object, create a Session object for working with Domino objects, and access the Domino Directory. In this case, we proceed only if the database was found.

```
try
{
    NotesThread.sinitThread();
    Session s = NotesFactory.createSession();
    Database db = s.getDatabase("", "names.nsf");
    if (db != null)
    {
```

Then we create a new document, set the form value for the new document, and populate the fields of the new document with the values we've retrieved.

```
Document doc = db.createDocument();
doc.replaceItemValue("Form", "Person");
doc.replaceItemValue("FirstName", fname);
doc.replaceItemValue("LastName", lname);
doc.replaceItemValue("OfficePhoneNumber", phone);
doc.replaceItemValue("MailAddress", email);
```

We then save the document and release the system resources used by the document, database, and session objects. (Before the session object is released, an error message is specified for cases where the database was not found.)

```
        doc.save(true);
        doc.recycle();
        db.recycle();
    }
    else
    {
        pw.println("An error was encountered.");
    }
    s.recycle();
}
```

Finally, we handle the Domino-related exceptions, terminate the thread, and display a message on the browser.

```
        catch (NotesException n)
        {
            pw.println("Notes Error: " + n.id);
            pw.println("Description: " + n.text);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        finally
        {
            NotesThread.stermThread();
        }
        pw.println(fname + " " + lname + " has been registered.");
        pw.println("</body></html>");
    }
}
```

The servlet code is only half the picture. Let's turn our attention to the setup of the Domino form. The form must be directed to the servlet, so you must

add a new HTML form tag with an action specifying the servlet. This routes the form to the servlet upon submission. A closing form tag must be placed before this tag; this closes the Domino-generated form tag. Here is the design of the form:

Notes.Net Example

First Name:

Last Name:

Email:

Telephone:

Submit

Objects | Reference

Submit (Button) : onClick

Run JavaScript

document.NotesNetExample.submit();

Here is the HTML, which appears at the top of the form, that redirects the form to the servlet:

```
[</form>
<form name="NotesNetExample" action="
http://baseline2000:81/servlet/Example_3">]
```

The brackets ([ and ]) designate the text as pass-thru HTML. The name attribute of the form element assigns a name to the form. This name is used when accessing form elements in JavaScript or the submission of the form to the server. The action attribute signals where the action is transferred when submitted

The following screen shows the form opened in Internet Explorer. The form data is sent to the servlet when it is submitted.

Notes.Net Example

First Name:

Last Name:

Email:

Telephone:

Submit

Address: http://baseline2000:81/notesnet.nsf/NotesNetExample?OpenForm

## Conclusion

Making the choice between a servlet and an agent is not a clear-cut decision. Language isn't a barrier since both support Java. Each has advantages, but it is important that Domino supports both. The decision is placed where it should be—on the developer. You may prefer servlets due to your extensive Web development background or because you want to use the powerful servlet manager in WebSphere. On the other hand, experienced Domino developers may prefer the robust Domino agent environment. The choice is yours.

## ABOUT THE AUTHOR

Anthony Patton works with various technologies like Java, XML, HTML, and Domino. He is the author of "Practical LotusScript" and "Domino Development With Java", both available from [Manning Publications](#). You can reach him via e-mail at [asp01@aye.net](mailto:asp01@aye.net).