# Using the Web Application Services with Microsoft Active Server Pages

The Web Application Services Toolkit is an extension to the Jolt 1.1 Java class library. The Toolkit allows the Jolt client class library to be used in a Web Server (such as Netscape Enterprise Server or Microsoft Active Server) to provide an interface between HTML clients or browsers, and a TUXEDO application.

A complete listing of all the examples used in this chapter are distributed with the Jolt Web Application Services software. In this chapter, segments of code from these samples are used to illustrate the use of the Toolkit. The samples delivered with the software supports four services: INQUIRY, WITHDRAWAL, DEPOSIT, and TRANSFER. This chapter explains the steps you can follow to use an HTML client interface to the TRANSFER service of the TUXEDO bankapp application. The TRANSFER service illustrates the use of parameters with multiple occurrences. This walkthrough explains the use of the TRANSFER service only.

**Note:** The walkthrough illustrates the use of the Web Application Services with Microsoft IIS and VBScript.

To use the information in the following sections, you should be familiar with:

- BEA TUXEDO and the sample TUXEDO application, bankapp
- BEA Jolt 1.1

- HTML (Hypertext Markup Language)
- VB Script
- object-oriented programming concepts

# Overview

Follow these steps to complete the Web Application Services walkthrough:

- Review the Getting Started Checklist
- Review the Overview of the TRANSFER Service
- Complete the Steps in the TRANSFER Request walkthrough
  - Initializing the Jolt Session Pool Manager
  - Submitting a TRANSFER Request from the Client
  - Processing the Request
  - Returning the Results to the Client

# Getting Started Checklist

Review this checklist before starting the TRANSFER Request Walkthrough.

**Note:** This checklist applies to Microsoft Active Server Pages only.

1. Ensure that you have a supported browser installed on your client machine. The client machine must have a network connection to the Web server that is used to connect to the TUXEDO environment.
2. Configure and boot TUXEDO and the TUXEDO bankapp example.
   a. Make sure the TRANSFER service is available.

b. Refer to the BEA TUXEDO user documentation for information about completing this task.

3. Refer to the *BEA Jolt User's Guide* Volume 1.1 for information about how to configure a Jolt Server.

   a. Note the *hostname* and *port number* associated with your Jolt Server Listener (JSL).

   b. Ensure that the TRANSFER service is defined in the Jolt Repository.

   c. Test the TRANSFER service using the Jolt Repository Editor to make sure it is accessible to Jolt clients.

4. Make sure you have Microsoft IIS 4.0 up and running.

   a. Check that script execution permission is enabled in the Web server application properties.

   b. Refer to the user documentation that accompanies the Microsoft IIS server for instructions.

5. Install the Jolt Web Application Services classes as described in the README file. This file is located in the directory where you extracted the Jolt Web Application Services software. The README file describes how to make these files available to the Web server.

   **Note:** There are several README files provided with this software. Make sure you are reading the correct file located in the directory `<extract_directory>/JoltWAS_iis`.

6. Install the teller sample application. Follow the instructions in the README file provided with this sample application. The location of this README is `<extract_directory>/JoltWAS_iis/teller`.

   **Note:** Edit the file, `web_start.inc`, to specify the correct *hostname* and *port number* for the Jolt Session Pool.

7. The code samples shown in "TRANSFER Request Walkthrough" are available from a sample application delivered with the Jolt Web Application Services software. Table 1 lists the files in the sample application. These files are a valuable reference for the walkthrough and are located in `<extract_directory>/teller`.

**Table 1 Bankapp Sample Source Files**

| File Name | Description |
|---|---|
| `tellerForm.asp` | Initializes the Jolt Session Pool Manager and displays available bankapp services. |
| `transferForm.htm` | Presents an HTML form for user input. |
| `tlr.asp` | Processes the HTML form and returns results as an HTML page. |
| `web_admin.inc` | VBScript functions for initializing the Jolt Session Pool Manager. |
| `web_start.inc` | VBScript functions for initializing the Jolt Session Pool Manager. |
| `web_templates.inc` | VBScript functions for caching HTML templates. |
| `templates/transfer.temp` | HTML templates used for returning results. |

# Overview of the TRANSFER Service

The TRANSFER Service in bankapp moves funds between two accounts. The service takes two account numbers, an input amount, and returns two balances—one for each account. In addition, the service returns an error message if there is an application or system error.

A TRANSFER is a WITHDRAWAL and a DEPOSIT executed as a single transaction. The transaction is created on the server, so the client does not need to create a transaction.

The client interface consists of an HTML page with a form used to enter the required data — account numbers and a dollar amount. This data is sent to the Web server as a `"POST"` request.

In the Web server, this request is processed using a VBScript Active Server Page. This program extracts the input data fields from the request, formats them for use with the Jolt Web Application Services class library, and dispatches the request to the TRANSFER service in the bankapp application. The TRANSFER service returns the results of the transaction. These results are returned to the VBScript program that merges them into a dynamically created HTML page. This page is returned to the client via the Web server infrastructure.

In the final part of this walkthrough, run the necessary HTML pages and server-side VBScript logic to execute a TRANSFER.

# TRANSFER Request Walkthrough

This section explains what happens when you execute a TRANSFER request. Each step is not illustrated here, only those steps that are necessary.

Included are:

- Initializing the Jolt Session Pool Manager
- Submitting a TRANSFER Request from the Client
- Processing the Request
- Returning the Results to the Client

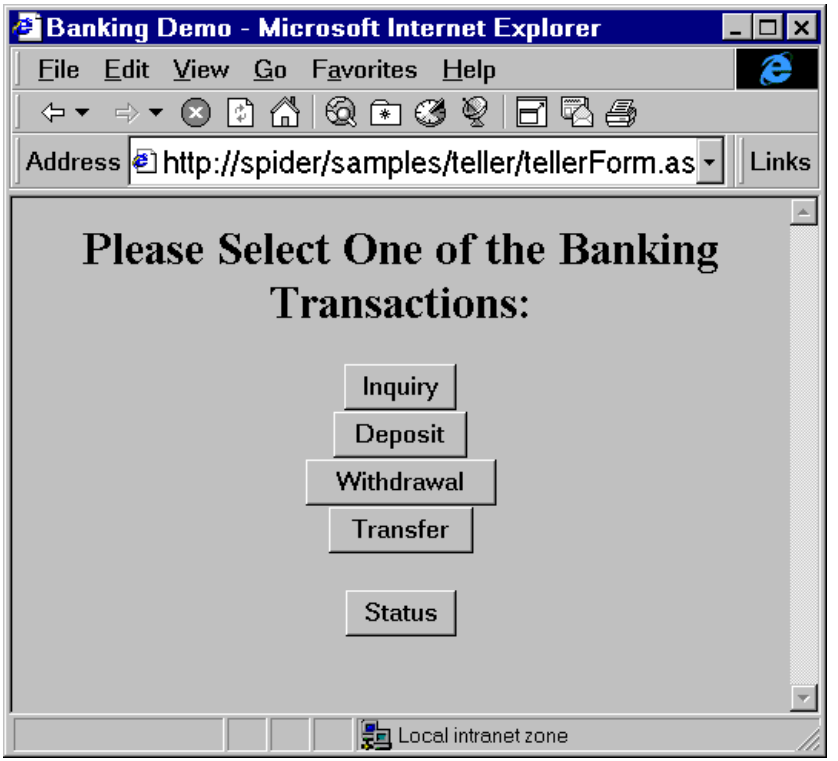
## Initializing the Jolt Session Pool Manager

To start the walkthrough, use the browser on your client to connect to the Web server where the Jolt Web Application Services classes are installed. The first page to download is `tellerForm.asp` (see Figure 1 for an example of a `tellerForm.asp` page). If the teller sample has been installed as described in step 6 of the "Getting Started Checklist," the URL for this page will be:

`http://<web-server:port>/teller/tellerForm.asp`

**Note:** The use of the port number is optional, depending on how your Web server is configured. In most cases, you are not required to add the ":port" in the URL.

**Figure 1 tellerForm.asp Example**




The page, `tellerForm.asp` contains VBScript procedures required to initialize the Jolt Session Pool Manager. The initialization code is contained in an ASP Script block. This code tells the Web server to execute this block of code on the server, instead of sending it to the client.

**Listing 1 tellerForm.asp: Initialize the Jolt Session Pool Manager**

```
<%
'// Initialize the session manager and cache templates
Call web_initSessionMgr(Null)
Call web_cacheTemplates()
%>
```

The VBScript procedure `web_initSessionMgr()` calls other VBScript procedures to establish a pool of Jolt Sessions. A Jolt Session is established between the Jolt Web Application Services in the Web Server and the Jolt Servers that reside in your TUXEDO application. One of the procedures called is `web_start()`. This procedure (in the file `web_start.inc`) should have been edited as part of the teller application installation process in step 6 of the "Getting Started Checklist".

The procedure `web_cacheTemplates()` reads various HTML template files into a memory cache. This step is not required, but it improves performance.

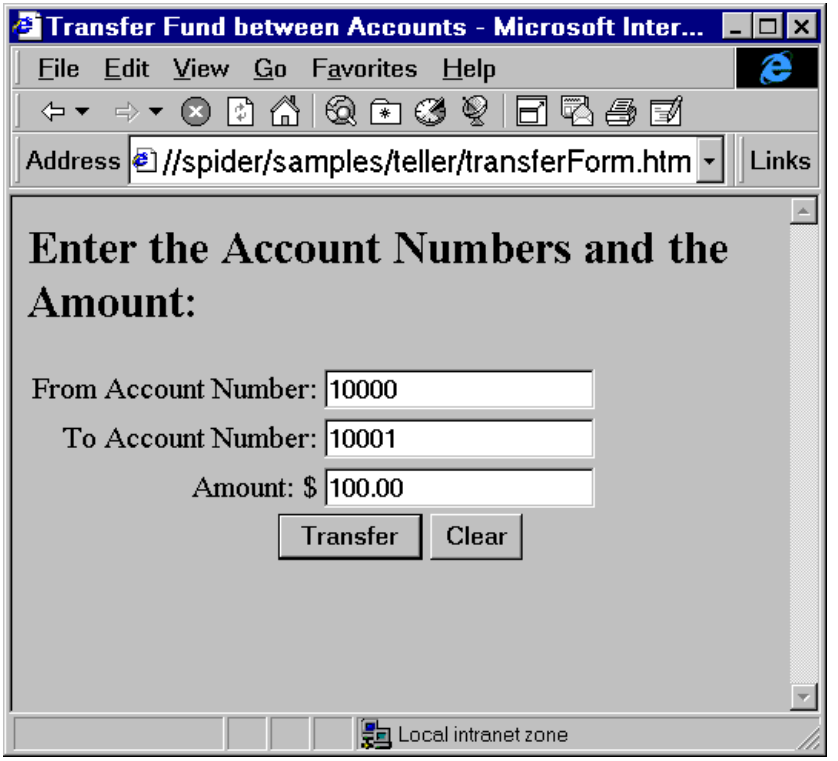
**Listing 2 tellerForm.asp: Allow the user to choose TRANSFER service**

```
<INPUT TYPE="button" VALUE="Transfer"
    onClick="window.location='transferForm.htm'">
```

The HTML segment above displays a button labeled Transfer. When this button is selected, the browser loads the page `transferForm.htm`. This page presents a form used to enter the data required by the TRANSFER service.

# Submitting a TRANSFER Request from the Client

**Figure 2 transferForm.htm Example**




The form in Figure 2 is generated by the page `transferForm.htm`. This page presents you with a form that will be used for input. The page consists of three text fields (two account numbers and a dollar amount), and a button that, when pressed, will cause the TRANSFER service to be invoked.

The code segment in Listing 3 shows the key HTML elements for this page. The **highlighted** elements in Listing 3 correspond to the elements in Table 2.

**Listing 3 transferForm.htm: TRANSFER Form**

```
<FORM NAME="teller" ACTION="tlr.asp" METHOD="POST">
<TABLE>
<TR><TD ALIGN=RIGHT>From Account Number: </TD>
    <TD><INPUT TYPE="text" NAME="ACCOUNT_ID_0"></TD></TR>
<TR><TD ALIGN=RIGHT>To Account Number: </TD>
    <TD><INPUT TYPE="text" NAME="ACCOUNT_ID_1"></TD></TR>
<TR><TD ALIGN=RIGHT>Amount: $</TD>
    <TD><INPUT TYPE="text" NAME="SAMOUNT"></TD></TR>
</TABLE>
<CENTER>
<INPUT TYPE="hidden" NAME="SVCNAME" VALUE="TRANSFER">
<INPUT TYPE="submit" VALUE="Transfer">
<INPUT TYPE="reset" VALUE="Clear">
</CENTER>
</FORM>
```

**Table 2 Key HTML Elements and Descriptions**

| Element | Description |
|---|---|
| `ACTION="tlr.asp"` | When the "submit" button is pressed, the contents of this form are delivered to a page called `tlr.asp` on the Web server for processing. |
| `NAME="ACCOUNT_ID_0"` | Shows the use of a field with multiple occurrences. The TRANSFER service expects two input account numbers, both called "ACCOUNT_ID". By using a convention of appending an *underscore* and *occurrence_number* (e.g., _0, _1) to the field name, both the name of a field and its occurrence can be passed to the program on the Web Server. |
| `NAME="SAMOUNT"` | Shows the use of an input field that has a single occurrence. In this example, there is nothing appended to the name of the field. |

The HTML form field names used in this example exactly match the TUXEDO field names expected by the TRANSFER service. This is not required, but doing so facilitates processing on the server because you do not have to map these inputs to TUXEDO field names. This is done by the Jolt Web Application Services classes.

The hidden field SVCNAME is assigned a value of TRANSFER. This field does not appear on the client form, but it is sent to the Web server as part of the request. The VBScript program retrieves the value of this field in order to determine which TUXEDO service is to be called (in this example, the service is TRANSFER).

Complete the fields `From Account Number`, `To Account Number`, and `Amount`. (10000 and 10001 are valid bankapp account numbers*).* Press the "Transfer" button. The data entered on the form is sent to the Web server for processing by the program `tlr.asp` as specified in the ACTION field of the form.

# Processing the Request

When the Web server receives the TRANSFER request, it runs the program `tlr.asp`. Client requests are turned into a Request object in the Web server. This Request object has members containing all the data that was input into the form along with other form data, such as hidden fields, etc. The Web server makes the Request object available to the program being invoked.

The program `tlr.asp` contains only VBScript. The first action performed by this program verifies that the Jolt Session Pool Manager is initialized. The code example shown in Listing 4 performs the initialization check and returns an HTML error page if the pool is not initialized.

**Listing 4 tlr.asp: Verify the Jolt Session Pool Manager is Initialized**

```
<%
If Not IsObject(Application("mgr")) Then
%>
    <HTML>
    <HEAD><TITLE>Error</TITLE></HEAD>
    <BODY><CENTER>
    <H2>Session Manager is not initialized</H2>
    <P>Make sure that you access the correct HTML
    </CENTER></BODY>
    </HTML>
```

```
<%
End If
%>
```

If the session pool is initialized, the program continues to process the request. The program locates a Session from the Session Pool Manager shown in Listing 5.

**Listing 5 tlr.asp: Locate a session**

```
Set pool = Application("mgr").getSessionPool(Null)
```

Once a valid session is located, the program retrieves an HTML template that is used to return the results to the client. In this example, these templates were cached in the initialization section. The template retrieved is identified by the name of the service being invoked, `Request("SVCNAME")` shown in Listing 6.

**Listing 6 tlr.asp: Retrieve a Cached HTML Template**

```
'// Choose the response template
If IsEmpty(Application("templates")) Then
   Set template = Server.CreateObject("BEAWEB.Template")
Else
   Select Case Request("SVCNAME")
      Case "INQUIRY"
         Set template = Application("templates")(INQUIRY)
      Case "DEPOSIT"
         Set template = Application("templates")(DEPOSIT)
      Case "WITHDRAWAL"
         Set template = Application("templates")(WITHDRAWAL)
      Case "TRANSFER"
         Set template = Application("templates")(TRANSFER)
   End Select
End If
```

Next, call the TUXEDO service. In this example, the input data from the request object is passed to the `call()` method of the session. The `call()` method uses the built-in ASP Request object as input. The results of the `call()` are stored in the `output` object and an array, `iodata`.

**Listing 7 tlr.asp: Invoke the TUXEDO Service**

```
Set output = pool.call(Request("SVCNAME"), Null, Nothing)
Set iodata(1) = output
```

After you invoke the TUXEDO service, the `output` object and the second element of the array `iodata` contain the results of the service call.

**Note:** In this example, because the initial form specified field names match the TUXEDO service parameter names, the Request object can be used in the `call()` method. If these names do not match, create an input array with "name=value" elements for each service parameter before invoking the `call()` method.

# Returning the Results to the Client

At this stage, no results have been returned to the client. The final step sends an HTML page containing the results of the service call back to the client. The HTML page consists of the template merged with the data returned by the service call shown in Listing 7.

The template file contains placeholders for variable (call specific) data. These placeholders are identified by the special tag *<%=NAME%>*. In the code example shown in Listing 8, an index is used to indicate which occurrence of a parameter name is used. For example, `ACCOUNT_ID[0]` specifies the first occurrence of the field `ACCOUNT_ID`.

**Listing 8 transfer.temp: Placeholders for TRANSFER Results**

```
<TABLE BORDER=1>
<TR><TD></TD><TD ALIGN=CENTER><B>Account #</B></TD>
   <TD ALIGN=CENTER><B>Balance</B></TR>
```

```
<TR><TD ALIGN=RIGHT><B>From:</B></TD><TD><%=ACCOUNT_ID[0]%></TD>
   <TD><%=SBALANCE[0]%></TR>
<TR><TD ALIGN=RIGHT><B>To:</B></TD><TD><%=ACCOUNT_ID[1]%></TD>
   <TD><%=SBALANCE[1]%></TR>
</TABLE>
```

To substitute the placeholders in the template with the actual values of the data returned from the service call, use the `eval()` method of the Template object shown in Listing 9. This method matches placeholders in the template file with fields of the same name in the results data and replaces them accordingly. A check for valid results (`output` object) is done as shown in Listing 9. If there is no output object, an error template page is returned.

**Listing 9 tlr.asp: Template Processing**

```
path = Application("templatedir")
If (Not IsObject(output)) Or (output is Nothing) Then
    Call template.evalFile(path & "\nosession.temp", Null)
Elseif output.noError() Then
    Call template.eval(iodata)
Elseif output.applicationError() Then
    Call template.evalFile(path & "\error.temp", iodata)
Else
    '// System error
    Dim errdata(0)
    Set errdata(0) = Server.CreateObject("BEAWEB.TemplateData")
    Call errdata(0).setValue("ERRNO", output.getError())
    Call errdata(0).setValue("ERRMSG", output.getStringError())
    Call template.evalFile(path & "\syserror.temp", errdata)
End If
```

**Note:** The array *iodata* contains both the input request and the results from the service call. This is useful if you want the results page to contain data that is part of the input.

When the template is processed, the resulting HTML is returned to the client as shown in Figure 3.

**Figure 3 tlr.asp Results Page**