BEA Jolt

Web Applicaton Services Developer's Guide

Jolt Release 1.1
Document Edition 1.0
May 1998

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Jolt Web Application Services Developer's Guide

Document Edition	Date	Software Version
1.0	May 1998	Jolt Release 1.1

Contents

3. Using the Web Application Services Toolkit

4. Web Application Services Class Library Reference

5. BEAWEB Components for Microsoft Active Server Pages

Preface

Purpose of This Document

This document describes the BEA Jolt Web Application Services product and gives instructions for using the tools for creating applications.

Who Should Read This Document

This document is intended for developers who are interested in building applications using a Web Server scripting language and Jolt. It assumes a familiarity with BEA Jolt, BEA TUXEDO, JavaScript or VB Script, and Java programming.

How This Document Is Organized

The *BEA Jolt Web Application Services Developer's Guide* is organized as follows:

- Chapter 1, "Introducing the Jolt Web Application Services," describes the Jolt Web Application Services features and how they work.
- Chapter 2, "Installing the Jolt Web Application Services," describes the requirements for installing the Jolt Web Application Services.
- Chapter 3, "Using the Web Application Services Toolkit," describes how developers use the Jolt Web Application services to create applications.
- Chapter 4, "Web Application Services Class Library Reference," is a reference of the Jolt Web Application Services class library methods and classes.

How to Use This Document

This document, *BEA Jolt Web Application Services Developer's Guide*, is designed primarily as an online, hypertext document. If you are reading this as a paper publication, note that to get full use from this document you should install and access it as an online document via a Web browser.

The following sections explain how to view this document online, and how to print a copy of this document.

Opening the Document in a Web Browser

To access the online version of this document, open the following HTML file in a Web browser:

```
/joltwasdoc/index.htm
```

Note: The online documentation requires a Web browser that supports HTML version 3.0. Netscape Navigator version 2.02 or Microsoft Internet Explorer version 3.0 or later are recommended.

Figure 1 shows the online document with the clickable navigation bar and table of contents.

Figure 1 Online Document Displayed in a Netscape Web Browser

Printing from a Web Browser

You can print a copy of this document, one file at a time, from the Web browser. Before you print, make sure that the chapter or appendix you want is displayed and *selected* in your browser. (To select a chapter or appendix, click anywhere inside the chapter or appendix you want to print. If your browser offers a Print Preview feature, you can use the feature to verify which chapter or appendix you are about to print.)

The BEA Jolt Online Documentation Home Page also includes Adobe Acrobat PDF files of all of the online documents. You can use the Adobe Acrobat Reader to print all or a portion of each document.

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys sequentially.
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

Convention	Item
`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Examples*: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

`|`	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

`...`	Indicates one of the following in a command line:
	♦ That an argument can be repeated several times in a command line
	♦ That the statement omits additional optional arguments
	♦ That you can enter additional parameters, values, or other information
	The ellipsis itself should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

Related Documentation

The following sections list the documentation provided with the Jolt software, and other publications related to Jolt and TUXEDO technology.

Jolt Documentation

The Jolt information set consists of the following documents:

BEA Jolt User's Guide

BEA Jolt Release Notes

Note: The BEA Jolt Online Documentation Home Page also includes Adobe Acrobat PDF files of all of the online documents. You can use the Adobe Acrobat Reader to print all or a portion of each document.

BEA Publications

The following BEA publications are also available:

TUXEDO System Reference Manual

TUXEDO System Programmer's Guide, Volumes 1 and 2

Contact Information

The following sections provide information about how to obtain support for the documentation and software.

Documentation Support

If you have questions or comments on the documentation, you can contact the BEA Information Engineering Group by e-mail at **docsupport@beasys.com** or by telephone at **+1.408.542.4193**. (For information about how to contact Customer Support, refer to the following section.)

Customer Support

If you have any questions about this version of BEA Jolt Add-ons, or if you have problems installing and running BEA Web Application Services, contact BEA Customer Support through BEA WebSupport at `www.beasys.com`. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

CHAPTER

1 Introducing the Jolt Web Application Services

The Jolt 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 the Netscape Enterprise Server or Microsoft Internet Information Server) to provide an interface between HTML clients or browsers, and TUXEDO services.

The Jolt Web Application Services provides an ease-of-use interface for processing and generating dynamic HTML pages with the Jolt Web Application Services. You do not need to learn how to write Common Gateway Interface (CGI) transactional programs to access TUXEDO services.

Netscape Enterprise Server 3.01 and the Microsoft Internet Information Server 4.0 are the supported Web servers for the Jolt Web Application Services.

The following topics are discussed in this chapter:

- Key Features
- How the Jolt Web Application Services Works

Key Features

The Jolt Web Application Services, an extension to the Jolt class library, enables TUXEDO services and transactions to be invoked from a Web server using a scripting language.

Some of the benefits of this architecture include:

- The HTML interface is preserved.
- The need to download Java class files is eliminated along with the delays associated with the download.
- Session Pooling efficiently utilizes the TUXEDO resources.
- Leverages industry standard HTTP protocol with encryption, and firewall configuration for the Web server.

Note: Asynchronous notification is not available in the Web Application Services. It is recommended that Jolt enabled Java clients (applets) be written using a retained connection to support asynchronous notification.

Refer to Chapter 3, "Using the Web Application Services Toolkit," for information about the how to use the Jolt Web Application services and complete a sample walkthrough.

How the Jolt Web Application Services Works

The Jolt Web Application Services architecture includes three main components: a session, a session pool, and a session pool manager. A **session** object represents a connection with the TUXEDO system. A **session pool** represents many physical connections between the Web server and the TUXEDO system. It also associates a session with an HTTP request.

The **session pool manager** is responsible for maintaining a set of session objects, each having a unique session identifier.

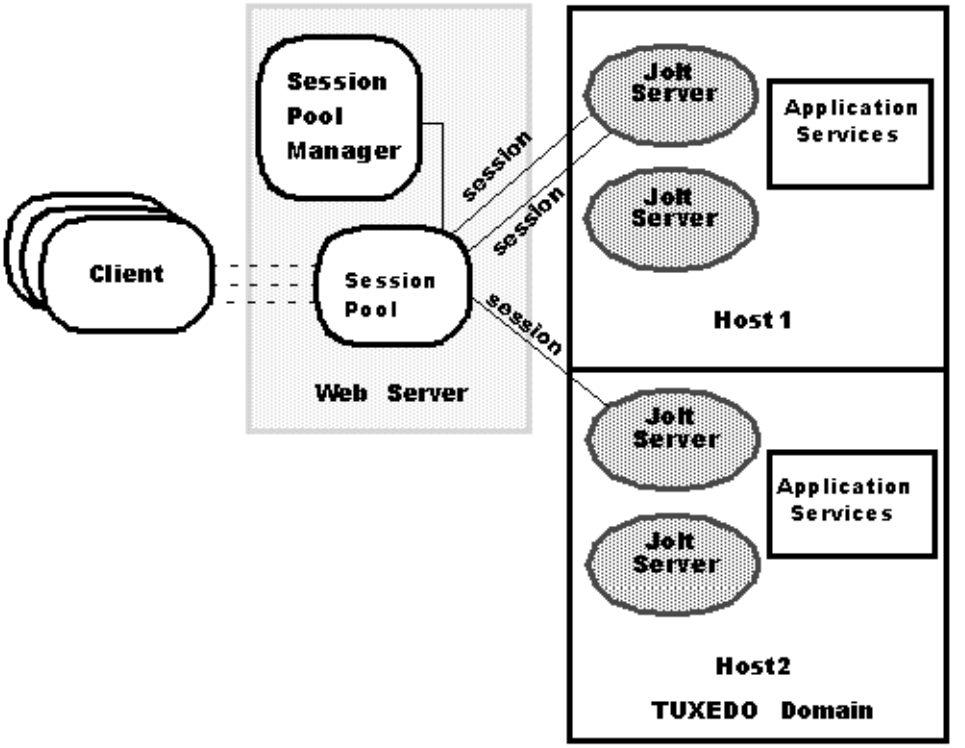
1. If the Web application has not be initialized the Web Application initializes the **session pool manager**, creates a **session pool**, and establishes **sessions** (also known as connections) with the Jolt Server.

2. When a service request arrives, the Web application gets a **session pool** object from the **session pool manager**. The **session pool** invokes the service call using the **session** that is the "least busy," based on the number of outstanding call requests on a given session.

3. If the selected **session** is terminated by the Jolt server, the **session pool** object restarts a new session or reroutes the request to another session. If the **session pool manager** is unable to get any session, a null session object is returned.

A graphical representation of the Web Application Services architecture is shown in Figure 1-1.

Figure 1-1 Jolt Web Application Services Architecture



Refer to the Chapter 4, “Web Application Services Class Library Reference,” for additional information about the SessionPool Class and SessionPoolManager Class.

Refer to Chapter 3, “Using the Web Application Services Toolkit,” for information about the how to use the Jolt Web Application services and complete a sample walkthrough.

2 Installing the Jolt Web Application Services

The chapter outlines the requirements for installing Jolt Web Application Services and bankapp services, and includes the following sections:

- Netscape Enterprise Server Requirements
- Microsoft Active Server Requirements
- Web Application Services Class Files Installation
  - Netscape Enterprise Server Samples
  - Microsoft Information Services Server Samples

Netscape Enterprise Server Requirements

The following components are required before installing the Jolt Application Services on the Netscape Enterprise Server:

Note: Currently, Netscape Enterprise Server 3.5.1 is not supported.

- Netscape Enterprise Server 3.0.1
- TUXEDO 6.x
- Jolt 1.1 Patch 5 (or higher) — contact BEA Customer Support for this patch

Microsoft Active Server Requirements

The following components are required before installing Jolt Application Services on the Microsoft Active Server:

- Windows NT Server 4.0 with NT Option Pack
- Microsoft IIS 4.0
- TUXEDO 6.x
- Jolt 1.1 Patch 5 (or higher) — contact BEA Customer Support for this patch

Refer to the online Microsoft NT Option Pack product documentation, specifically the Microsoft Internet Information Server chapters.

Web Application Services Class Files Installation

Follow the directions in the `<extract_directory>/<compressed_file>/README` to install the Jolt Web Application Services class files into your Web server. Where `<compressed_file>` is the name of the tar or self extracting zip file without the extension, `JoltWAS_NS` for Netscape Enterprise Server and `JoltWAS_IIS` for Microsoft Information Services Server.

Netscape Enterprise Server Samples

The following three sample directories are shipped with the Jolt Web Application Service Netscape Enterprise Server product, teller, bankapp, and service. Both the teller and bankapp samples use the TUXEDO bankapp services, so TUXEDO must have been configured and booted and the TUXEDO bankapp example running before using the samples.

The directions for installing and configuring the teller sample application services can be found in the `<extract_directory>/JoltWAS_NS/teller/README` file. The Jolt Web Application Services teller sample application illustrates the TUXEDO BankApp using the server-side JavaScript built-in request object in the service call, and shows the template processing capabilities. This sample shows the coupling methods which map the members of the request object to TUXEDO fields.

The directions for installing and configuring the bankapp sample application services can be found in the `<extract_directory>/JoltWAS_NS/bankapp/README` file. The Jolt Web Application Services bankapp sample illustrates the API for the TUXEDO BankApp, the use of transactions on the Web server, and the template processing.

Microsoft Information Services Server Samples

The teller sample directory is shipped with the Jolt Web Application Service Microsoft Information Services Server product. This sample uses the TUXEDO bankapp services, so TUXEDO must have been configured and booted and the TUXEDO bankapp example running before using the sample.

The directions for installing and configuring the teller sample application services can be found in the `<extract_directory>/JoltWAS_IIS/teller/README` file. The Jolt Web Application Services teller sample application illustrates the TUXEDO BankApp using the server-side JavaScript built-in request object in the service call, and shows the template processing capabilities. This sample shows the coupling methods which map the members of the request object to TUXEDO fields.

3 Using the Web Application Services Toolkit

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 Toolkit 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 provide support 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 Netscape Enterprise Server. If you are using the Microsoft IIS and VBScript, you need to use the corresponding Visual Basic methods.

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)

- JavaScript (if you are using Netscape Enterprise Server)
- VB Script (if you are using Microsoft IIS)
- object-oriented programming concepts

Note: With Netscape Enterprise Server, the interface to Java classes is through LiveConnect and server-side JavaScript. Microsoft Internet Information Server uses Active Server Pages and VBScript as its primary interface.

This walkthrough describes the use of the Jolt Web Application Services Toolkit with Netscape Enterprise Server and server-side JavaScript. The last section of this chapter shows the VB Script (Microsoft IIS) equivalent.

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 Netscape Enterprise Server 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 Netscape Enterprise Server 3.0.1 up and running.
   a. Check that Java and server side JavaScript are enabled in the Web server.
   b. Refer to the user documentation that accompanies the Netscape browser 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_ns`.
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_ns/teller`.

   Note: Edit the file, `web_start.js` 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 3-1 lists the files in the sample application. These files are a valuable reference for the walkthrough and are located in `<extract_directory>/teller`.

Table 3-1 Bankapp Sample Source Files

File Name	Description
`tellerForm.html`	Initializes the Jolt Session Pool Manager and displays available bankapp services.
`transferForm.html`	Presents an HTML form for user input.
`tlr.html`	Process the HTML form and returns results as an HTML page.
`web_admin.js`	JavaScript functions for initializing the Jolt Session Pool Manager.
`web_start.js`	JavaScript functions for initializing the Jolt Session Pool Manager.
`web_templates.js`	JavaScript 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 server-side JavaScript program. 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 JavaScript 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 JavaScript 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

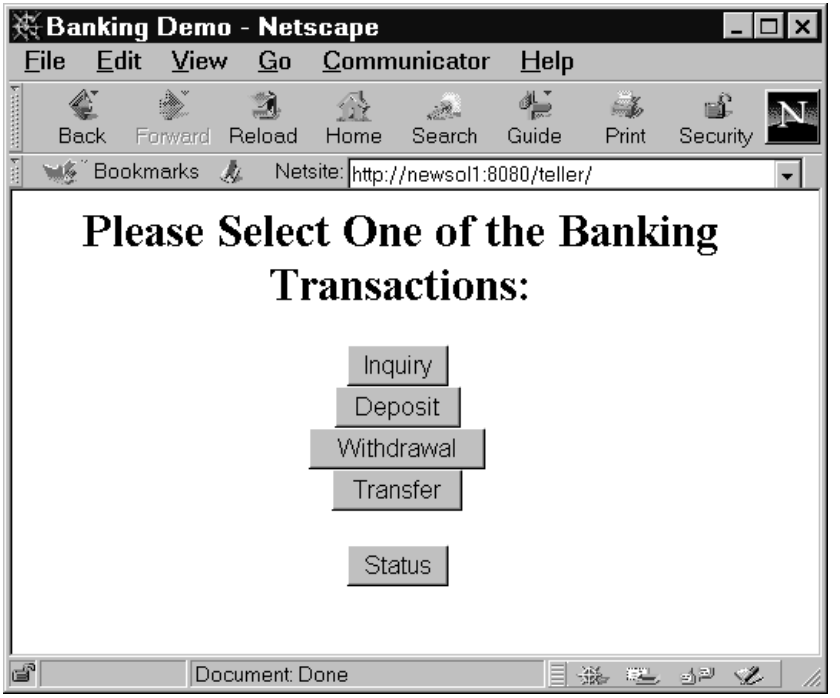
To start the walkthrough, use the browser on your client to connect to the Web server where the Jolt Web Application Services classes are installed. The first page to download is `tellerForm.html`. If the teller sample has been installed as described in step 6 of the "Getting Started Checklist," the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

`http://<web-server:port>/teller`

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 3-1 is an example of the `tellerForm.html` page.

Figure 3-1 tellerForm.html Example



The page, `tellerForm.html` contains JavaScript functions required to initialize the Jolt Session Pool Manager. The initialization code is contained in a `<SERVER>..</SERVER>` block. This code tells the Web server to execute this block of code on the server, instead of sending it to the client.

Listing 3-1 tellerForm.html: Initialize the Jolt Session Pool Manager

```
<SERVER>
// Initialize the session manager and cache templates.
web_initSessionMgr(null);
web_cacheTemplates();
</SERVER>
```

The JavaScript function `web_initSessionMgr()` calls other JavaScript functions to establish a Jolt Session. The 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 functions called is `web_start()`. This function (in the file `web_start.js`) should have been edited as part of the teller application installation process in step 6 of the "Getting Started Checklist".

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

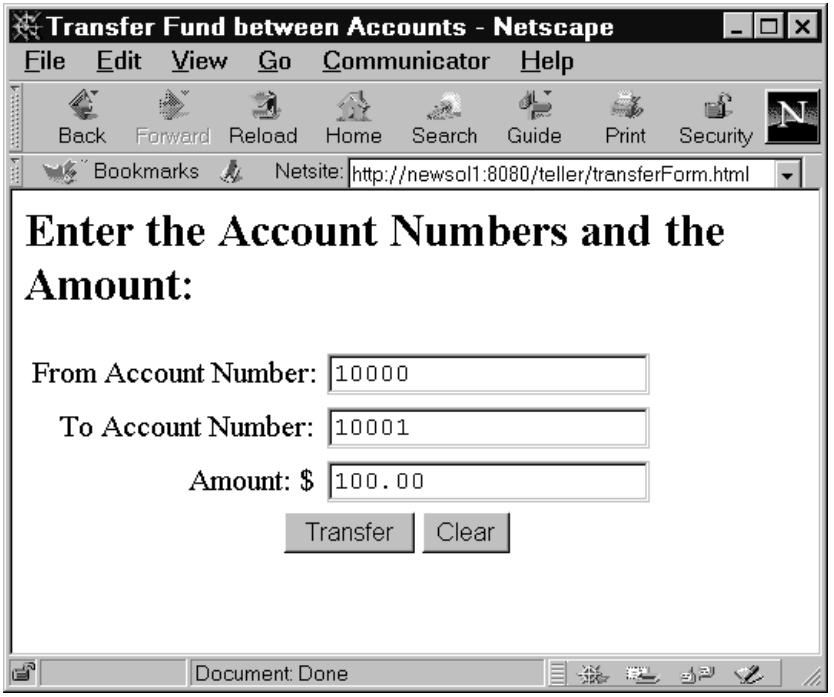
Listing 3-2 tellerForm.html: Allow the user to choose TRANSFER service

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

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

Submitting a TRANSFER Request from the Client

Figure 3-2 transferForm.html Example



The form in Figure 3-2 is generated by the page `transferForm.html`. 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 3-3 shows the key HTML elements for this page. The **highlighted** elements in Listing 3-3 correspond to the elements in Table 3-2.

Listing 3-3 transferForm.html: TRANSFER Form

```
<FORM NAME="teller" ACTION="tlr.html" 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>
<INPUT TYPE="hidden" NAME="SVCNAME" VALUE="TRANSFER">
<INPUT TYPE="submit" VALUE="Transfer">
<INPUT TYPE="reset" VALUE="Clear">
</FORM>
```

Table 3-2 Key HTML Elements and Descriptions

Element	Description
`ACTION="tlr.html"`	When the "submit" button is pressed, the contents of this form are delivered to a page called `tlr.html` 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 a 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 server-side JavaScript 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`. (Account numbers 10000 and 10001 are usually valid in bankapp*)*. Press the "Transfer" button. The data entered on the form is sent to the Web server for processing by the program `tlr.html` 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.html`. Client requests are turned into a request object in the Web server. This request object has members containing all the data which 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.html` contains only server-side JavaScript. The first action performed by this program verifies that the Jolt Session Pool Manager is initialized. The code example shown in Listing 3-4 performs the initialization check and returns an HTML error page if the pool is not initialized.

Listing 3-4 tlr.html: Verify the Jolt Session Pool Manager is Initialized

```
if (project.mgr == null) {
       write("<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>");
}
```

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 3-5.

Listing 3-5 tlr.html: Locate a session

```
var session = project.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 3-6.

Listing 3-6 tlr.html: Retrieve a Cached HTML Template

```
// Template
        var path = project.templatedir;
        var template = project.templates[request.SVCNAME];
        if (template == null)
                template = new Packages.bea.web.ns.Template();
```

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 accepts the request object as a parameter. The results of the `call()` are stored in the `output` object and an array, `iodata`.

Listing 3-7 tlr.html: Invoke the TUXEDO Service

```
var output = iodata[1] = session.call(request.SVCNAME, request);
```

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 passed unchanged into the `call()` method. If these names do not match, manipulate the request object 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 3-7.

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 3-8, use an index 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 3-8 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 class shown in Listing 3-9. 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 3-9. If there is no output object, an error template page is returned.

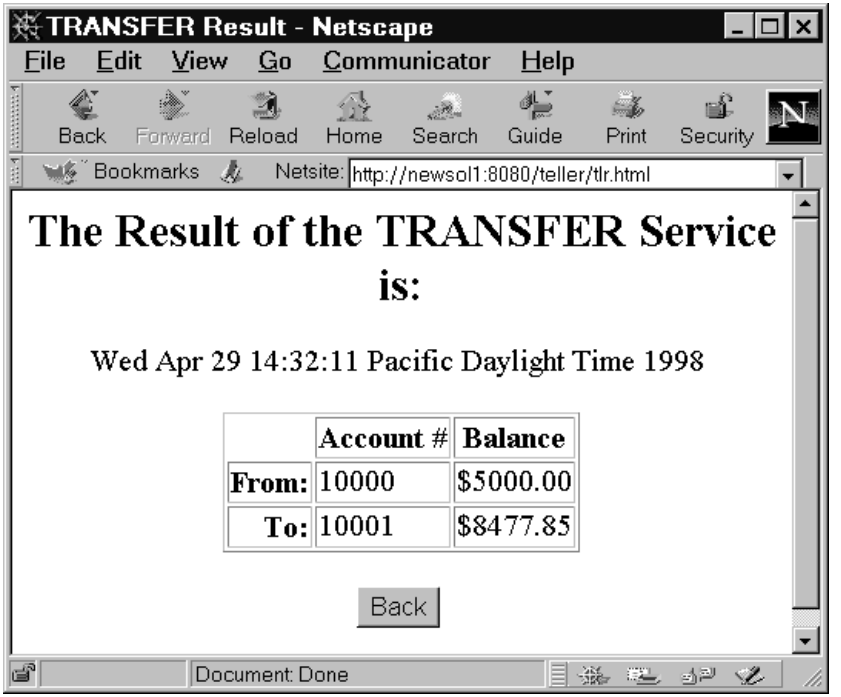
Listing 3-9 tlr.html: Template Processing

```
if (output == null)
       template.evalFile(path+"templates/nosession.temp", null);
else if (output.noError())
       template.eval(iodata);
```

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 3-3.

Figure 3-3 tlr.html Results Page



4 Web Application Services Class Library Reference

The Web Application Services Class Library consists of object-oriented Java language classes for creating a TUXEDO session.

The Web Application Services Class Library Reference includes the following topics:

- bea.web Package
  - UserInfo Class
  - SessionPoolManager Class
  - SessionPool Class
  - Connection Class
  - Transaction Class
  - Template Class
  - TemplateData Class
  - Result Class
- bea.web.ns Package
  - SessionPoolManager Class
  - SessionPool Class
  - Template Class
  - TemplateData Class

bea.web Package

This Java package contains all the core Jolt wrapper classes that are neutral to all Web servers. These classes are:

- UserInfo Class
- SessionPoolManager Class
- SessionPool Class
- Connection Class
- Transaction Class
- Template Class
- TemplateData Class
- Result Class

UserInfo Class

```
java.lang.Object
   |
   +----bea.web.UserInfo
```

This class contains the TUXEDO user authentication information. This information may be needed to create a TUXEDO session. If the user does not want to expose the authentication information on the web server, the user may embed the authentication information within a derived class.

UserInfo Constructor

The following constructor creates an instance of the UserInfo class.

UserInfo

This constructor creates an instance of the UserInfo class.

Synopsis **`UserInfo`**`()`

UserInfo Methods

The following methods are used in conjunction with the UserInfo class:

- setUserName
- setUserPassword
- setUserRole
- setAppPassword

setUserName

Synopsis `void` **`setUserName`**`(String` *`username`*`)`

Parameters *`username`*
Specifies a valid user name.

Usage You must set the TUXEDO user name before creating a session pool. This is only required if the user authorization level is specified in TUXEDO.

setUserPassword

Synopsis `void` **`setUserPassword`**`(String` *`password`*`)`

Parameters *`password`*
A valid user password.

Usage You must set the TUXEDO user password before creating a session. This is only required if the user authorization level is specified in TUXEDO.

setUserRole

Synopsis `void **setUserRole**(String *userrole*)`

Parameters *userrole*
A valid user role.

Usage You must set the TUXEDO user role before creating a session. This is only required if the user authorization level is specified in TUXEDO.

setAppPassword

Synopsis `void **setAppPassword**(String *password*)`

Parameters *password*
A valid application password.

Usage You must set the TUXEDO application password before creating a session. This is only required if the application or user authorization level is specified in TUXEDO.

SessionPoolManager Class

```
java.lang.Object
   |
   +----java.util.Dictionary
           |
           +----java.util.Hashtable
                    |
                    +----bea.web.SessionPoolManager
```

This base class represents a session pool manager which manages one or more session pools. Each session pool identified by an ID contains a pool of connections to Jolt servers.

When an HTTP request arrives, the session pool manager uses an ID to find the corresponding session pool to serve the request.

SessionPoolManager Methods

The following methods are used in conjunction with the SessionPoolManager class:

- done
- createSessionPool
- stopSessionPool
- getSessionPool
- suspendSessionPool
- removeSessionPool

done

Synopsis `final void done()`

Usage Done with this session manager.

createSessionPool

Synopsis

```
final int createSessionPool(String addrs[], String saddrs[],
int minCons, int maxCons, UserInfo usr, String id)
```

Parameters

addrs
A list of primary server addresses.

saddrs
A list of secondary server addresses for failover.

minCons
The minimum number of connections to start with.

maxCons
The maximum number of connections in this pool.

usr
The TUXEDO authentication information, or null.

id
Reserved for future use. Must be null.

Usage

Create a session pool with multiple restartable connections. The caller provides a unique ID to identify the session pool. If a connection in the session pool is terminated unexpectedly, a new connection will be established to keep the session pool alive. A session pool provides load-balancing among connections and failover capability by letting the user specify a primary server address list and a secondary server address list. The *id* must be null for this release.

Returns

This method returns the number of successful connections.

stopSessionPool

Synopsis

```
final void stopSessionPool(String id)
```

Parameters

id
Reserved for future use. Must be null.

Usage

Shut down a session pool; all active connections in the session are also terminated immediately. The *ID* must be null for this release. This method is part of the administrative class library.

getSessionPool

Synopsis `final SessionPool **getSessionPool**(String *id*)`

Parameters *id*
Reserved for future use. Must be null.

Usage Get the session pool. If the session pool does not exist, this method will return a null object. The *ID* must be null for this release.

Returns A session pool or null if it does not exist.

suspendSessionPool

Synopsis `final void **suspendSessionPool**(String *id*, boolean *action*)`

Parameters *id*
Reserved for future use. Must be null.

action
`true` to suspend the session pool; `false` to resume it.

Usage Suspend this session pool so it will not accept any new requests. It allows the administrator to shut down the session pool gracefully. The *ID* must be null for this release. This method is part of the administrative class library.

removeSessionPool

Synopsis `final void removeSessionPool(String *id*)`

Parameters *id*
Removed for future use. Must be null.

Usage Shut down and remove the session pool; all active connections in this session pool are terminated immediately. The *ID* must match the *id* in `createSessionPool()`. This method is part of the administrative class library.

SessionPool Class

```
java.lang.Object
   |
   +----bea.web.SessionPool
```

This base class represents a session pool that contains one or more connections (sessions) to Jolt servers. The session pool is responsible to select the least busy connection to serve each service call. An application may tie multiple service calls into a transaction. To start a transaction, the session pool selects the least busy connection and puts it in transactional mode.

Unless the number of connections in the session pool is one or the request is engaged in a transaction, there is no guarantee that the service calls in each HTTP request will be served by the same connection. All service calls associated with the transaction are handled by the same connection. The connection is locked until the transaction is committed or aborted. The transaction should be committed or aborted before continuing to another page, unless programmatic arrangements to commit or abort the transaction have been made.

SessionPool Methods

The following methods are used in conjunction with the SessionPool class:

- startTransaction
- getConnection
- getMaxConnections
- setMaxConnections
- isSuspended

startTransaction

Synopsis `Transaction` **`startTransaction`**`(int` *`timeout`*`)`

Parameters *`timeout`*
The transaction time-out value in seconds.

Usage Start a transaction. The transaction should be committed or aborted before continuing to another page, unless programmatic arrangements to commit or abort the transaction have been made. Otherwise, the connection is locked until the transaction is terminated.

Returns A transaction object, or null if failure occurs.

getConnection

Synopsis `int` **`getConnection`**`(int` *`index`*`)`

Usage Get a connection. This method is part of the administrative class library.

Parameters *`index`*
A connection index.

getMaxConnections

Synopsis `int` **`getMaxConnections`**`()`

Usage Get the maximum number of connections configured in this session. This method is part of the administrative class library.

setMaxConnections

Synopsis `int` **`setMaxConnections`**`(int` *`maxCons`*`)`

Parameters *`maxCons`*
The new session pool size.

Usage Change the session pool size. Currently shrinking the session pool is disallowed. This method is part of the administrative class library.

Returns The new spool size, or -1 for shrinking the session pool.

isSuspended

Synopsis `boolean **isSuspended**()`

Usage Check if this session pool is suspended. A suspended session pool does not accept any new requests. This method is part of the administrative class library.

Connection Class

```
java.lang.Object
   |
   +----bea.web.Connection
```

This class represents a connection (session) to a Jolt server and provides simple methods for service invocations and transaction demarcation. This class is a part of the administrative class library.

Connection Methods

The following methods are used in conjunction with the Connection class:

- getAccessTime
- getAddr
- getErrorCount
- getUseCount
- inTransaction
- isAlive

getAccessTime

Synopsis `final Date` **`getAccessTime`**`()`

Usage Get the last access time of this object.

getAddr

Synopsis `final String` **`getAddr`**`()`

Usage Get the server address.

getErrorCount

Synopsis `final int **getErrorCount**()`

Usage Get the cumulative system error count.

getUseCount

Synopsis `final int **getUseCount**()`

Usage Get the outstanding request count.

inTransaction

Synopsis `boolean **inTransaction**()`

Usage Check if the connection is in a transaction.

Returns `True` if the connection is in a transaction; `false` otherwise.

isAlive

Synopsis `boolean **isAlive**()`

Usage Check if the connection is still alive.

Returns `True` if the connection is still alive; `false` otherwise.

Transaction Class

```
java.lang.Object
   |
   +----bea.web.Transaction
```

This class is a wrapper for the bea.joltJoltTransaction class. If an error occurs, the method in this class returns an error code instead of throwing an exception.

Transaction Methods

The following methods are used in conjunction with the Transaction class:

- commit
- rollback

commit

Synopsis `int commit()`

Usage Commit the transaction.

Returns 0 if successful; otherwise, a TUXEDO error number.

rollback

Synopsis `int rollback()`

Usage Roll back the transaction.

Returns 0 if successful; otherwise, a TUXEDO error number.

Template Class

```
java.lang.Object
   |
   +----bea.web.Template
```

This base class processes a template and replaces tags of the form <%=*name*%> or indexed tags of the form <%=*name*[*index*]%> (where *name* and *index* are user-defined constructs) with values. *name* must start with an underscore (_) or letter and be followed by one or more letters, digits, or underscores. The values from the TemplateData Class are used to populate the tags. If a duplicate name exists in the multiple TemplateData objects, the later element has a higher precedence.

Template Methods

The following methods are used in conjunction with the Template class:

- load
- unload

load

Synopsis `final int` **`load`**`(String` *`path`*`)`

Parameters *path*
The path to the template file.

Usage Load the template file into memory. The contents are cached until the `unload` method is called.

Returns 0 if success; -1 if file cannot be opened.

unload

Synopsis `final void` **`unload`**`()`

Usage Unload the cached template contents.

TemplateData Class

```
java.lang.Object
   |
   +----java.util.Dictionary
           |
           +----java.util.Hashtable
                    |
                    +----bea.web.TemplateData
```

This base class extends from `java.util.Hashtable` and it provides a fast mechanism to store and retrieve a data element by name and index. The keys in the hash table are the names of the data elements.

TemplateData Methods

The following methods are used in conjunction with the TemplateData class:

- setValue — by index
- setValue
- getValue — by index
- getValue
- getBytesValue — by index
- getBytesValue
- setBytesValue — by index
- setBytesValue
- getCount

setValue – by index

Synopsis `void **setValue**(String *name*, int *index*, String *value*)`

Parameters

name
The name of the data element.

index
The index of the data element.

value
The value of the data element.

Usage Set the string value of a data element associated with a name and an index.

setValue

Synopsis `void **setValue**(String *name*, String *value*)`

Parameters

name
The name of the data element.

value
The value of the data element.

Usage This method is the same as `setValue(*name*,0,*value*)` for a non-octet value.

getValue – by index

Synopsis `String **getValue**(String *name*, int *index*, String *defaultValue*)`

Parameters

name
The name of the data element.

index
The index of the data element.

defaultValue
The default value if the element does not exist.

Usage Get the string value of a data element associated with a name and an index. Most output data, except `carray` (octets) data type, are automatically converted to String. If there is a typecast error, the default value will be returned.

Returns The value of the data element or the default value.

getValue

Synopsis `String` **`getValue`**`(String` *`name`*`, String` *`defaultValue`*`)`

Parameters *`name`*
The name of the data element

`defaultValue`
The default value if the element does not exist

Usage This method is same as `getValue(`*`name`*`,0,`*`defaultValue`*`)`.

Returns The value of the data element, or the default value.

getBytesValue — by index

Synopsis `byte[] getBytesValue(String` *`name`*`, int` *`index`*`, byte` *`defval`*`[])`

Parameters *`name`*
The name of the data element.

`index`
The index of the data element, starting from 0.

`defval`
The default value if the data element does not exist.

Usage Get the octet value of a data element associated with a name and an index. If an output field has `carray` (octets) data type, the data remains as `byte[]`. The caller must use this method to retrieve the data. If there is a typecast error, the default value will be returned.

Returns The octet value of the data element or the default value.

getBytesValue

Synopsis `byte[] getBytesValue(String` *`name`*`, byte` *`defval`*`[])`

Parameters *`name`*
The name of the data element.

`defval`
The default value if the attribute does not exist.

Usage This method is same as `getBytesValue(name, 0, defval)` method.

Returns The octet value of the data element or the default value.

setBytesValue — by index

Synopsis `void setBytesValue(String *name*, int *index*, byte *value*[])`

Parameters

name
The name of the data element.

index
The index of the data element.

value
The octet value for the data element.

Usage Set the octet value of a data element associated with a name and an index. The data element should be defined as `carray` in TUXEDO.

setBytesValue

Synopsis `void setBytesValue(String *name*, byte *value*[])`

Parameters

name
The name of the data element.

value
The octet value for the data element.

Usage This method is same as `setBytesValue(name,0,value)` for the octet value.

getCount

Synopsis `int **getCount**(String *name*)`

Parameters

name
The name of the data element.

Usage Get the occurrence count of a data element.

Returns The count.

Result Class

```
java.lang.Object
   |
   +----java.util.Dictionary
           |
           +----java.util.Hashtable
                    |
                    +----bea.web.TemplateData
                             |
                             +----bea.web.Result
```

This class represents the result set from the service invocation. Most output data types are converted into `String`, except `carray` which remains as `byte[]`. The caller must use the `getBytesValue()` methods to get the `carray` data. The keys to the hash table are the field names in the result set. Since this class is derived from the TemplateData class, it can be used by the Template class for template processing.

Result Methods

The following methods are used in conjunction with the Result class:

- applicationError
- getApplicationCode
- getError
- getErrorDetail
- getStringError
- getStringErrorDetail
- noError
- systemError

applicationError

Synopsis `boolean applicationError()`

Usage Test if an application error occurred in the last service invocation. The application error is equivalent to `TPESVCFAIL` in TUXEDO.

Returns `True` if an application error occurred; `false` otherwise.

getApplicationCode

Synopsis `int getApplicationCode()`

Usage Get the application code returned by the service. The application code is same as the `tpurcode` in TUXEDO.

Returns The application code.

getError

Synopsis `int getError()`

Usage Get the system (non-`TPESVCFAIL`) error number. The error number is the same as the `tperrno` in TUXEDO.

Returns The error number.

getErrorDetail

Synopsis `int getErrorDetail()`

Usage Get the detail system error number. This information is available only to TUXEDO 6.4 and then only if the service call has timed out.

Returns The detail system error number, if TUXEDO 6.4; otherwise 0.

getStringError

Synopsis `String getStringError()`

Usage Get the system error message.

Returns The system error message. If it is not a system error, null is returned.

getStringErrorDetail

Synopsis `String getStringErrorDetail()`

Usage Get the detail system error message. This information is available only to TUXEDO 6.4 and then only if the service call has timed out.

Returns The detail system error message, if TUXEDO 6.4; otherwise null.

noError

Synopsis `boolean noError()`

Usage Test whether no error was encountered in the last service invocation.

Returns `True` if no error occurred; `false` otherwise.

systemError

Synopsis `boolean systemError()`

Usage Test whether a system error occurred in the last service invocation.

Returns `True` if a system error occurred; `false` otherwise.

bea.web.ns Package

The `bea.web.ns` package contains the classes specific for the Netscape server-side JavaScript. These classes are:

- SessionPoolManager Class
- SessionPool Class
- Template Class
- TemplateData Class

SessionPoolManager Class

```
java.lang.Object
   |
   +----java.util.Dictionary
           |
           +----java.util.Hashtable
                   |
                   +----bea.web.SessionPoolManager
                           |
                           +----bea.web.ns.SessionPoolManager
```

This class provides a session pool factory for the Netscape-specific `SessionPool` object and provides the conversion from LiveWire `JSObject` to Jolt object. The actual implementation of the session pool manager is in its parent class.

SessionPoolManager Constructor

The following constructor creates an instance of the SessionPoolManager class.

SessionPoolManager

This constructor creates an instance of the SessionPoolManager class.

Synopsis

```
SessionPoolManager()
```

SessionPoolManager Method

The following method is used in conjunction with the SessionPoolManager class:

- createSessionPool

createSessionPool

Synopsis

```
int createSessionPool(JSObject addrs, JSObject saddrs,
int minCons, int maxCons, UserInfo usr, String id)
```

Parameters

addrs
A JavaScript Array of primary server addresses.

saddrs
A JavaScript Array of secondary fail-over server addresses, or null.

minCons
The minimum number of connections.

maxCons
The maximum number of connections.

usr
The user authentication information, or null.

id
A unique ID for the session pool, or null.

Usage

Create a session pool with multiple restartable connections. This method is the same as `bea.web.createSessionPool`, except the *addrs* and *saddrs* parameters are JavaScript Array objects (for example, `addrs[0] = "//mach:7000"`).

Returns

The number of successful connections.

SessionPool Class

```
java.lang.Object
   |
   +----bea.web.SessionPool
            |
            +----bea.web.ns.SessionPool
```

This class extends from the bea.web.SessionPool and it provides an additional call() method that understands Netscape server-side JavaScript object.

SessionPool Methods

The following methods are used in conjunction with the SessionPool class:

- call
- call — with transaction

call

Synopsis `Result` **`call`**`(String` *`name`*`, JSObject` *`indata`*`)`

Parameters

`name`
The name of a TUXEDO service.

`indata`
JavaScript string Array or built-in request object

Usage Call a service without involving any transaction. This is equivalent to using the `call(name, indata, null)` method.

Returns A Result object.

call — with transaction

Synopsis

```
Result call(String name, JSObject indata, Transaction tran)
```

Parameters

name
The name of a TUXEDO service.

indata
JavaScript string Array or built-in request object.

tran
A Transaction object.

Usage

Call a service with a transaction. The *indata* parameter is either a JavaScript Array of name-value pairs or a server-side JavaScript Request object. (A name-value pair is a construct of the form *name=value*.) The name in the name-value pair should match the parameter name in the Jolt service definition. The member name in the Request object can have a trailing underscore and an index number (as, for example, `SBALANCE_0`) if it is a multiple occurrence field. Any nonmatching names in the *indata* parameter are ignored. The *tran* parameter must be the object returned by the `startTransaction()` method.

Returns

A Result object.

Template Class

```
java.lang.Object
   |
   +----bea.web.Template
            |
            +----bea.web.ns.Template
```

This class extends from bea.web.Template and it provides some additional methods that understands Netscape server-side JavaScript objects.

Template Constructors

The following constructor creates an instance of the Template class.

Synopsis **`Template()`**

Usage A default constructor.

Template Methods

The following methods are used in conjunction with the Template class:

- eval — data set array
- eval — data set
- evalFile — data set array
- evalFile — data set

eval — data set array

Synopsis
```
int eval(JSObject data)
```

Parameters

data

An array of TemplateData objects.

Usage
Process the cached template with a list of data set and send the processed template to the client. The list must be a JavaScript Array object.

Results
0 if success; -1 for any I/O error.

eval — data set

Synopsis
```
final int eval(TemplateData data)
```

Parameters

data

A template data set, or null.

Usage
Process the cached template with a data set and send the processed template to the client.

Returns
0 if success; -1 for any I/O error.

evalFile — data set array

Synopsis
```
int evalFile(String path, JSObject data)
```

Parameters

path

An absolute path of the template file

data

An array of TemplateData objects.

Usage
Process a template file with a list of data sets. The list must be a JavaScript Array object.

Returns
0 if success; -1 if the template file cannot be opened or there is an I/O error.

evalFile — data set

Synopsis `final int evalFile(String path, TemplateData data)`

Parameters

path
A path of the template file.

data
A template data set, or null.

Usage Process a template file with a data set and send the processed template to the client.

Returns 0 if success; -1 if the template file cannot be opened or there is an I/O error.

TemplateData Class

```
java.lang.Object
   |
   +----java.util.Dictionary
           |
           +----java.util.Hashtable
                    |
                    +----bea.web.TemplateData
                             |
                             +----bea.web.ns.TemplateData
```

This class represents the data set that can be used by template processing or service invocation to store the result set. It provides an extra method to import the server-side JavaScript request object. The keys to the hash table are the field names in the data set.

TemplateData Constructors

The following constructor creates an instance of the TemplateData class.

Synopsis `TemplateData()`

Usage A default constructor.

TemplateData Methods

The following method is used in conjunction with the TemplateData class:

- importRequest

importRequest

Synopsis `void` **`importRequest`**`(JSObject` *`request`*`)`

Parameters *`request`*
A server-side JavaScript Request object.

Usage Import the values from the server-side Request into this data set object.

5 BEAWEB Components for Microsoft Active Server Pages

The BEAWEB components contain the objects specific for Microsoft Active Server Pages (ASP). These objects are:

- SessionPoolManager Object
- SessionPool Object
- Template Object
- TemplateData Object
- Connection Object
- UserInfo Object
- Transaction Object
- Result Object

SessionPoolManager Object

This object provides a session pool factory for the Microsoft specific SessionPool object.

BEAWEB.SessionPoolManager Component

The following component creates an instance of the SessionPoolManager object.

Synopsis `Server.CreateObject ("BEAWEB.SessionPoolManager")`

Usage Create a SessionPoolManager object.

SessionPoolManager Methods

The following methods are used in conjunction with the SessionPoolManager object:

- done
- createSessionPool
- suspendSessionPool
- stopSessionPool
- removeSessionPool
- getSessionPool

done

Synopsis **done**()

Usage Done with this session manager. Stops all session pools and terminates all active connections immediately.

createSessionPool

Synopsis

```
int createSessionPool(String addrs[], String saddrs[], int minCons,
int maxCons, UserInfo usr, String id)
```

Parameters

addrs
A list of primary server addresses.

saddrs
A list of secondary server addresses for failover.

minCons
The minimum number of connections to start with.

maxCons
The maximum number of connections in this pool.

usr
The TUXEDO authentication information, or null.

id
Reserved for future use. Must be null.

Usage

Create a session pool with multiple restartable connections. The caller provides a unique ID to identify the session pool. If a connection in the session pool is terminated unexpectedly, a new connection will be established to keep the session pool alive. A session pool provides load-balancing among connections and failover capability by letting the user specify a primary server address list and a secondary server address list. The id must be null for this release.

Returns

The number of successful connections.

suspendSessionPool

Synopsis

```
suspendSessionPool(String id, boolean action)
```

Parameters

id
Reserved for future use. Must be null.

action
`true` to suspend the session pool; `false` to resume it.

Usage

Suspend this session pool so it will not accept any new requests. It allows the administrator to shut down the session pool gracefully. The ID must be null for this release. This method is part of the administrative object library.

stopSessionPool

Synopsis `stopSessionPool(String id)`

Parameters *id*

Reserved for future use. Must be null.

Usage Shut down a session pool; all active connections in the session are also terminated immediately. The ID must be null for this release. This method is part of the administrative class library.

removeSessionPool

Synopsis `removeSessionPool(String id)`

Parameters *id*

Reserved for future use. Must be null.

Usage Shut down and remove the session pool; all active connections in this session pool are terminated immediately. The ID must be null for this release. This method is part of the administrative class library.

getSessionPool

Synopsis `SessionPool getSessionPool(String id)`

Parameters *id*

Reserved for future use. Must be null.

Usage Get the session pool. If the session pool does not exist, this method will return nothing. The ID must be null for this release.

Returns A session pool or nothing if it does not exist.

SessionPool Object

This object provides a SessionPool implementation for the Microsoft Active Server Pages environment. It is created by the createSessionPool method in the SessionPoolManager object.

SessionPool Methods

The SessionPool methods are used in conjunction with the SessionPool object:

- call
- startTransaction
- getConnection()
- setMaxConnections
- getMaxConnections
- isSuspended

call

Synopsis

```
Result call(String name, String indata[], Transaction tran)
```

Parameters

name
The name of a TUXEDO service.

indata
Array of name-value pairs or null for ASP built-in Request object.

tran
An optional Transaction object.

Usage

Call a service with a transaction (or null for no transaction). The *indata* is either a String array with name=value pairs or a null for an implicit ASP Request object. The name in the name=value pair should match the parameter name in the Jolt service definition. The member name in the Request object may have a trailing "_" and an

index number (e.g., SBALANCE_0) if it is a multiple occurrence field. Any non-matching names in the indata parameter or Request object are ignored. The tran parameter must be the object returned by the startTransaction() method.

Returns A Result object.

startTransaction

Synopsis `Transaction` **`startTransaction`**`(int` *`timeout`*`)`

Parameters *`timeout`*
The transaction time-out value in seconds.

Usage Start a transaction. The transaction should be committed or aborted before continuing to another page, unless programmatic arrangements to commit or abort the transaction have been made. Otherwise, the connection is locked until the transaction is terminated.

Returns A transaction object, or nothing if failure occurs.

getConnection()

Synopsis `Connection` **`getConnection`**`(int` *`index`*`)`

Parameters *`index`*
A connection index.

Usage Get a connection. This method is part of the administrative API.

setMaxConnections

Synopsis `int setMaxConnections(int` *`maxCons`*`)`

Parameters *`maxCons`*
The new session pool size.

Usage Change the session pool size. Currently shrinking the session pool is disallowed. This method is part of the administrative class library.

Returns The new pool size.

getMaxConnections

Synopsis `int getMaxConnections()`

Usage Get the maximum number of connections configured in this session pool. This method is part of the administrative class library.

isSuspended

Synopsis `boolean isSuspended()`

Usage Check if this session pool is suspended, so it does not accept any new requests. This method is part of the administrative class library.

Template Object

This object provides the Template implementation for the Microsoft Application Server Pages. The output of the template content is sent to the web server response buffer.

BEAWEB.Template Component

The following component creates an instance of the Template object.

Synopsis
```
Server.CreateObject ("BEAWEB.Template")
```

Template Methods

The following methods are used in conjunction with the Template object:

- eval — data set list
- evalFile
- load
- unload

eval — data set list

Synopsis
```
int eval(TemplateData data[])
```

Parameters

data

An array of TemplateData or Result objects.

Usage Process the cached template with a list of data set and send the processed template to the response buffer. The list must be a VBScript Array object.

Results 0 if success; -1 for any error.

evalFile

Synopsis `int **evalFile**(String *path*, TemplateData *data*[])`

Parameters
path
A path of the template file

data
An array of TemplateData or Result objects

Usage Replace the tags in a template file with the values from a list of template data set. The list is an array object.

Returns 0 if success; -1 if the template file cannot be opened or there is an I/O error.

load

Synopsis `int **load**(String *path*)`

Usage Load the template from a file into memory for faster access. The contents are cached until the unload() is called.

Parameters
path
A path of the template file

Returns 0 if success, -1 if file cannot be opened.

unload

Synopsis `**unload**()`

Usage Unload the cached template.

TemplateData Object

This object represents the data set that can be used by template processing. It provides an extra method to import the ASP built-in Request object. The keys to the template data are the field names in the data set.

BEAWEB.TemplateData Component

The following component creates instances of the TemplateData object.

Synopsis

```
Server.CreateObject ("BEAWEB.TemplateData")
```

TemplateData Methods

The following methods are used in conjunction with the TemplateData object:

- setValue
- getValue
- setValueByIndex
- getValueByIndex
- setBytesValue
- getBytesValue
- getBytesValueByIndex
- setBytesValueByIndex
- getCount
- importRequest

setValue

Synopsis `setValue(String name, String value)`

Parameters

name
The name of the data element.

value
The string value for the data element.

Usage This method is same as `setValueByIndex(name,0,value)` for non-octet value.

getValue

Synopsis `String getValue(String name, String defval)`

Parameters

name
The name of the data element.

defval
The default string value for the data element, if it does not exist.

Usage Get the string value of a data element associated with a name. Most output data, except carray (octets) data type, are automatically converted to String. If there is a type-cast error, the default value will be returned.

Returns The string value of the data element or the default value.

setValueByIndex

Synopsis `setValueByIndex(String name, int index, String value)`

Parameters

name
The name of the data element.

index
The index of the data element, starting from 0.

value
The string value for the data element.

Usage Set the string value of a data element associated with a name and an index.

getValueByIndex

Synopsis `String` **`getValueByIndex`**`(String` *`name`*`, int` *`index`*`, String` *`defval`*`)`

Parameters

`name`
The name of the data element

`index`
The index of the data element, starting from 0.

`defval`
The default value if the data element does not exist.

Usage Get the string value of a data element associated with a name and an index. Most output data, except carray (octets) data type, are automatically converted to String. If there is a type-cast error, the default value will be returned.

Returns The string value of the data element or the default value.

setBytesValue

Synopsis **`setBytesValue`**`(String` *`name`*`, byte` *`value`*`[])`

Parameters

`name`
The name of the data element.

`value`
The byte value for the data element.

Usage This method is same as `setBytesByIndexValue(name,0,value)` for the octet value.

getBytesValue

Synopsis `byte[]` **`getBytesValue`**`(String` *`name`*`, byte` *`defval`*`[])`

Parameters

`name`
The name of the data element

`defval`
The default value if the data element does not exist.

Usage Get the octet value of a data element associated with a name. If an output field has carray (octets) data type, the data remains as byte[]. The caller must use this method to retrieve the data. If there is a type-cast error, the default value will be returned.

Returns The octet value of the data element or the default value.

getBytesValueByIndex

Synopsis
```
byte[] getBytesValueByIndex(String name, int index, byte defval[])
```

Parameters

name
The name of the data element

index
The index of the data element, starting from 0.

defval
The default value if the data element does not exist.

Usage
Get the octet value of a data element associated with a name and an index. If an output field has carray (octets) data type, the data remains as byte[]. The caller must use this method to retrieve the data. If there is a type-cast error, the default value will be returned.

Returns
The octet value of the data element or the default value.

setBytesValueByIndex

Synopsis
```
setBytesValueByIndex(String name, int index, byte value[])
```

Parameters

name
The name of the data element.

index
The index of the data element.

value
The byte value for the data element.

Usage
Set the octet value of a data element associated with a name and an index. The data element should be defined as carray in the TUXEDO application.

getCount

Synopsis
```
int getCount(String name)
```

Parameters

name
The name of the data element.

Usage
Get occurrence count of a data element.

Returns
The count of occurrences.

importRequest

Synopsis `void **importRequest**()`

Usage Import the values from the Microsoft ASP Request object into this TemplateData object.

Connection Object

This object represents a connection (TUXEDO session) to a Jolt server and provides simple methods for connection status. It is returned from the getConnection method on the SessionPool object. This object is a part of the administrative class library.

Connection Methods

The following methods are used in conjunction with the Connection object:

- getAccessTime
- getAddr
- getErrorCount
- getUseCount
- inTransaction
- isAlive

getAccessTime

Synopsis `Date getAccessTime()`

Usage Get the last access time of this object.

getAddr

Synopsis `String getAddr()`

Usage Get the server address.

getErrorCount

Synopsis `int getErrorCount()`

Usage Get the cumulative system error count.

getUseCount

Synopsis `int **getUseCount**()`

Usage Get the outstanding request count.

inTransaction

Synopsis `boolean **inTransaction**()`

Usage Check if the connection is in a transaction.

Returns `True` if the connection is in a transaction; `false` otherwise.

isAlive

Synopsis `boolean **isAlive**()`

Usage Check if the connection is still alive.

Returns `True` if the connection is still alive; `false` otherwise.

UserInfo Object

This object contains the TUXEDO user authentication information. This information may be needed to create a TUXEDO session.

BEAWEB.UserInfo Component

The following component creates an instance of the UserInfo object.

UserInfo

This component creates an instance of the UserInfo object.

Synopsis `Server.CreateObject ("BEAWEB.UserInfo")`

UserInfo Methods

The following methods are used in conjunction with the UserInfo object:

- setUserName
- setUserPassword
- setUserRole
- setAppPassword

setUserName

Synopsis **`setUserName`**`(String` *`username`*`)`

Parameters *`username`*
Specifies a valid user name.

Usage You must set the TUXEDO user name before creating a session pool. This is only required if the user authorization level is specified in TUXEDO.

setUserPassword

Synopsis **`setUserPassword`**`(String` *`password`*`)`

Parameters *`password`*
A valid user password.

Usage You must set the TUXEDO user password before creating a session. This is only required if the user authorization level is specified in TUXEDO.

setUserRole

Synopsis **`setUserRole`**`(String` *`userrole`*`)`

Parameters *`userrole`*
A valid user role.

Usage You must set the TUXEDO user role before creating a session. This is only required if the user authorization level is specified in TUXEDO.

setAppPassword

Synopsis **`setAppPassword`**`(String` *`password`*`)`

Parameters *`password`*
A valid application password.

Usage You must set the TUXEDO application password before creating a session. This is only required if the application or user authorization level is specified in TUXEDO.

Transaction Object

This object is a wrapper for the bea.jolt.JoltTransaction object. If an error occurs, the method in this object returns an error code instead of throwing an exception. This object is returned by the startTransaction method or the SessionPool object.

Transaction Methods

The following methods are used in conjunction with the Transaction object:

- commit
- rollback

commit

Synopsis `int commit()`

Usage Commit the transaction.

Returns 0 if successful; otherwise, a TUXEDO error number.

rollback

Synopsis `int rollback()`

Usage Roll back the transaction.

Returns 0 if successful; otherwise, a TUXEDO error number.

Result Object

This object represents the result set from the service invocation. Most output data types are converted into `String`, except `carray` which remains as `byte[]`. The caller must use the `getBytesValue()` methods to get the `carray` data. The keys the data in the Result object are the field names in the result set. Since this object is derived from the TemplateData object, it can be used by the Template object for template processing. This object is returned from the call() method or the SessionPool object.

Result Methods

The following methods are used in conjunction with the Result object:

- applicationError
- systemError
- getError
- getErrorDetail
- getApplicationCode
- getStringError
- getStringErrorDetail
- noError
- setValue
- getValue
- setValueByIndex
- getValueByIndex
- setBytesValue
- getBytesValue

- getBytesValueByIndex
- setBytesValueByIndex
- getCount

applicationError

Synopsis `boolean` **`applicationError`**`()`

Usage Test if an application error occurred in the corresponding service invocation. The application error is equivalent to `TPESVCFAIL` in TUXEDO.

Returns `True` if an application error occurs.

systemError

Synopsis `boolean` **`systemError`**`()`

Usage Test if a system error occurred in the corresponding service invocation.

Returns `True` if a system error occurs.

getError

Synopsis int **`getError`**()

Usage Get the system (non-`TPESVCFAIL`) error number. The error number is same as the `tperrno` in TUXEDO.

Returns The system error number.

getErrorDetail

Synopsis int **`getErrorDetail`**()

Usage Get the detail system error number. This information is available only to TUXEDO 6.4 and then only if the service call has timed out.

Returns The detail system error number or 0.

getApplicationCode

Synopsis: int **`getApplicationCode`**()

Usage: Get the application code returned by the service. The application code is same as the `tpurcode` in TUXEDO.

Returns: The application code.

getStringError

Synopsis: String **`getStringError`**()

Usage: Get the system error message. If it is not a system error, null is returned.

Returns: The system error message.

getStringErrorDetail

Synopsis: String **`getStringErrorDetail`**()

Usage: Get the detail system error message. This information is available only to TUXEDO 6.4 and then only if the service call has timed out.

Returns: The detail system error message or null.

noError

Synopsis: `boolean` **`noError`**`()`

Usage: Test if no error was encountered in the corresponding service invocation.

Returns: `True` if there is no error.

setValue

Synopsis: **`setValue`**`(String` *`name`*`, String` *`value`*`)`

Parameters:

`name`
The name of the data element.

`value`
The string value for the data element.

Usage: This method is same as `setValueByIndex(name,0,value)` for non-octet value.

getValue

Synopsis `String` **`getValue`**`(String` *`name`*`, String` *`defval`*`)`

Parameters *`name`*
The name of the data element.

`defval`
The default string value for the data element, if it does not exist.

Usage Get the string value of a data element associated with a name. Most output data, except carray (octets) data type, are automatically converted to String. If there is a type-cast error, the default value will be returned.

Returns The string value of the data element or the default value.

setValueByIndex

Synopsis **`setValueByIndex`**`(String` *`name`*`, int` *`index`*`, String` *`value`*`)`

Parameters *`name`*
The name of the data element.

`index`
The index of the data element, starting from 0.

`value`
The string value for the data element.

Usage Set the string value of a data element associated with a name and an index.

getValueByIndex

Synopsis
```
String getValueByIndex(String name, int index, String defval)
```

Parameters

name
The name of the data element

index
The index of the data element, starting from 0.

defval
The default value if the data element does not exist.

Usage Get the string value of a data element associated with a name and an index. Most output data, except carray (octets) data type, are automatically converted to String. If there is a type-cast error, the default value will be returned.

Returns The string value of the data element or the default value.

setBytesValue

Synopsis
```
setBytesValue(String name, byte value[])
```

Parameters

name
The name of the data element.

value
The byte value for the data element.

Usage This method is same as `setBytesByIndexValue(name,0,value)` for the octet value.

getBytesValue

Synopsis
```
byte[] getBytesValue(String name, byte defval[])
```

Parameters

name
The name of the data element

defval
The default value if the data element does not exist.

Usage Get the octet value of a data element associated with a name. If an output field has carray (octets) data type, the data remains as byte[]. The caller must use this method to retrieve the data. If there is a type-cast error, the default value will be returned.

Returns The octet value of the data element or the default value.

getBytesValueByIndex

Synopsis
```
byte[] getBytesValueByIndex(String name, int index, byte defval[])
```

Parameters

name
The name of the data element

index
The index of the data element, starting from 0.

defval
The default value if the data element does not exist.

Usage
Get the octet value of a data element associated with a name and an index. If an output field has carray (octets) data type, the data remains as byte[]. The caller must use this method to retrieve the data. If there is a type-cast error, the default value will be returned.

Returns
The octet value of the data element or the default value.

setBytesValueByIndex

Synopsis
```
setBytesValueByIndex(String name, int index, byte value[])
```

Parameters

name
The name of the data element.

index
The index of the data element.

value
The byte value for the data element.

Usage
Set the octet value of a data element associated with a name and an index. The data element should be defined as carray in the TUXEDO application.

getCount

Synopsis
```
int getCount(String name)
```

Parameters

name
The name of the data element.

Usage
Get occurrence count of a data element.

Returns
The count of occurrences.

importRequest

Synopsis `void **importRequest**()`

Usage Import the values from the Microsoft ASP Request object into this TemplateData object.