BEA WebLogic Enterprise

Using Java Enterprise Tuxedo

WebLogic Enterprise 5.1
Document Edition 5.1
May 2000

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Using Java Enterprise Tuxedo

Document Edition	Date	Software Version
5.1	May 2000	BEA WebLogic Enterprise 5.1

Contents

2. Invoking BEA Tuxedo Services

3. Configuring JET for Client Access

4. Using the Bulk Loader Program

5. Using the BEA Jolt Repository Editor

Index

About This Document

This document describes the Java Enterprise Tuxedo™ (JET) component of BEA WebLogic Enterprise™ (WLE), which enables Java servers running in the WebLogic Enterprise environment to access services in the BEA Tuxedo® environment.

This document includes the following topics:

- Chapter 1, “Introducing Java Enterprise Tuxedo,” introduces the JET architecture and provides an overview of how Java servers (CORBA/Java, EJB, and RMI servers) running in the WebLogic Enterprise environment can invoke BEA Tuxedo services using JET.
- Chapter 2, “Invoking BEA Tuxedo Services,” describes how to invoke a Tuxedo service from a Java server running in the WebLogic Enterprise environment using the JET Class Library.
- Chapter 3, “Configuring JET for Client Access,” describes how to configure JET in order to use the Bulk Loader program or the BEA Jolt Repository Editor.
- Chapter 4, “Using the Bulk Loader Program,” describes how to populate the Jolt Repository with BEA Tuxedo service definitions using the Bulk Loader utility.
- Chapter 5, “Using the BEA Jolt Repository Editor,” describes how to add, modify, delete, test, and export BEA Tuxedo service definitions in the Jolt Repository using the Jolt Repository Editor.

In addition, the following interoperability sample applications implement JET:

- CORBA/Java-to-Tuxedo Simpapp sample application
- EJB-to-Tuxedo Simpapp sample application

For more information about these sample applications, see *CORBA, J2EE, and Tuxedo Interoperability and Coexistence*.

What You Need to Know

This document is intended for Java programmers and system administrators who want to develop or support Java server applications that access Tuxedo services within the WebLogic Enterprise environment. It assumes that you are familiar with BEA Tuxedo and Java programming. You must also understand the details of any Tuxedo services that you want to invoke using JET.

e-docs Web Site

The BEA WebLogic Enterprise product documentation is available on the BEA Systems, Inc. corporate Web site. From the BEA Home page, click the Product Documentation button or go directly to the "e-docs" Product Documentation page at http://e-docs.bea.com.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

A PDF version of this document is available on the WebLogic Enterprise documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Enterprise documentation Home page, click the PDF Files button, and select the document you want to print.

If you do not have the Adobe Acrobat Reader installed, you can download it for free from the Adobe Web site at http://www.adobe.com/.

Related Information

For more information about CORBA, Java 2 Enterprise Edition (J2EE), BEA Tuxedo®, distributed object computing, transaction processing, C++ programming, and Java programming, see the WebLogic Enterprise *Bibliography* in the WebLogic Enterprise online documentation.

Contact Us!

Your feedback on the BEA WebLogic Enterprise documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Enterprise documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Enterprise 5.1 release.

If you have any questions about this version of BEA WebLogic Enterprise, or if you have problems installing and running BEA WebLogic Enterprise, contact BEA Customer Support through BEA WebSUPPORT at www.bea.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

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 simultaneously.
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames 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`

`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

Convention	Item
`{ }`	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.

CHAPTER

1 Introducing Java Enterprise Tuxedo

This topic includes the following sections:

- Key Features
- JET Workflow
- Key Components
- Comparison Between Jolt and JET

Java Enterprise Tuxedo™ (JET) is a Java-based application programming interface (API) to the BEA Tuxedo system for Java servers (CORBA/Java, EJB, and RMI servers) running in the WebLogic Enterprise (WLE) environment. JET is a Java class library and API that enables Java servers to invoke BEA Tuxedo services and process the results.

Key Features

With JET, you can leverage existing BEA Tuxedo services from a WebLogic Enterprise Java server application (CORBA/Java, EJB, or RMI servers). The key feature of the JET architecture is its simplicity. You can build, deploy, and maintain robust, modular, and scalable electronic commerce systems that operate over the Internet.

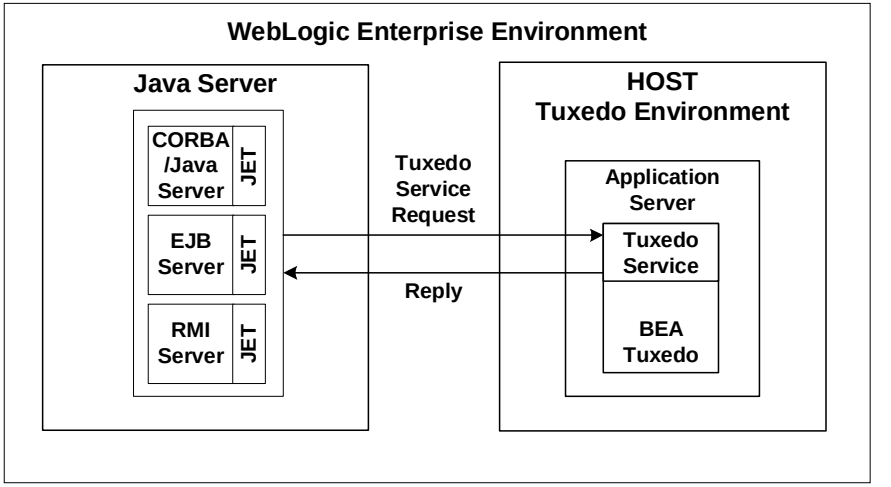
JET provides the following features:

- **Java-based API for simplified development—**with its Java-based API, JET simplifies application development by providing well-designed object interfaces into BEA Tuxedo services.
- **Implicit transaction support—**JET enables Java programmers to build Java servers that use the BEA Tuxedo application and transaction services without needing to understand detailed transactional semantics or rewrite existing BEA Tuxedo applications. Transactions are handled implictly by WebLogic Enterprise and BEA Tuxedo. Java programmers can decide whether a Java server application participates in, or is excluded from, a transaction.
- **BEA Tuxedo-enabled Java server development—**using JET, you can build Java servers that leverage your BEA Tuxedo services. Tuxedo processes can span multiple domains. JET calls can occur across a bridge or a domain gateway. JET provides the automatic conversion between Java and native BEA Tuxedo data types and buffers.
- **Easy access to BEA Tuxedo services through the Jolt Repository**—the Jolt Repository contains service definitions that represent Tuxedo services. JET provides graphical and command-line tools for dynamically populating and editing service definitions in the Jolt Repository.

JET Workflow

Figure 1-1 provides an overview of how JET works.

Figure 1-1 Java Servers Using JET to Invoke BEA Tuxedo Services



A BEA Tuxedo service invocation using JET involves the following steps:

1. A Java server (CORBA/Java, EJB, or RMI server) uses the API in the JET Class Library to prepare and submit a BEA Tuxedo service request.
2. JET retrieves the service definition for the requested BEA Tuxedo service and routes the request to the BEA Tuxedo server, which forwards the request to the Tuxedo service.
3. The BEA Tuxedo service receives the request and generates the results.
4. The BEA Tuxedo service returns the results to the Java server.
5. The Java server uses the JET Class Library to translate the results into a Java format and process the results.

This entire procedure occurs within the WebLogic Enterprise environment. For a more detailed description of this process, see Chapter 2, “Invoking BEA Tuxedo Services.”

Key Components

JET consists of the following key components:

- JET Class Library
- Jolt Repository Server
- Tools for Managing Service Definitions

Note: JET leverages technology from BEA Jolt®, a BEA product that links Web clients to BEA Tuxedo services. Certain JET components therefore include *Jolt* in the name. However, these components are used by JET, not Jolt. To use Jolt, you must purchase a Jolt license and install the Jolt software separately. For more information about Jolt, see "Comparison Between Jolt and JET" on page 1-16.

JET Class Library

The *JET Class Library* is a Java package that contains the class files that implement the JET API. These classes enable Java servers to invoke BEA Tuxedo services. The JET Class Library includes the functionality needed to prepare, submit, and process a BEA Tuxedo service request.

The following types of Java servers can use the JET Class Library:

- CORBA/Java servers
- EJB servers
- RMI servers

When developing a JET client application, you need to know only about the classes that JET provides and the BEA Tuxedo services that are defined in the Jolt Repository. JET hides the underlying application details. To use the JET Class Library, you do not need to understand the underlying transactional semantics, the language in which the services were coded, buffer manipulation, the location of services, or the names of

databases used. JET handles these operations for you and leverages the implicit transaction processing mechanisms that WebLogic Enterprise and BEA Tuxedo provide.

Table 1-1 describes the flow of activity involved in using the JET Class Library to access BEA Tuxedo services.

Table 1-1 Using the JET Class Library

Step	Process	Action
1	Import Library	Java server (CORBA/Java, EJB, or RMI server) imports the package containing the JET Class Library.
2	Prepare Request	Java server instantiates a `JoltService` object and prepares a BEA Tuxedo service request object using the JET API.
3	Submit Request	Java server invokes the `call` method to submit the service request.
4	Process Request	BEA Tuxedo application server receives and processes the service request.
5	Reply	BEA Tuxedo returns the results of the service invocation to the Java server.
6	Process Results	The Java server processes the results, including any exception handling.

For more information about calling Tuxedo services from within Java servers, see Chapter 2, “Invoking BEA Tuxedo Services.”

Jolt Repository Server

A BEA Tuxedo application is a collection of one or more services. The *Jolt Repository Server* (JREPSVR) is a Tuxedo server that manages service definitions of BEA Tuxedo services for JET. A *service definition* describes the properties of a BEA Tuxedo service, such as its name, input and output buffer types, and individual parameters.

JET uses these service definitions to perform data conversions. Tuxedo service definitions are stored in a central file, the *Jolt Repository*, which the JREPSVR manages. JET uses the JREPSVR to retrieve Tuxedo service definitions from the Jolt Repository. In addition, the JREPSVR manages updates to service definitions in the Jolt Repository.

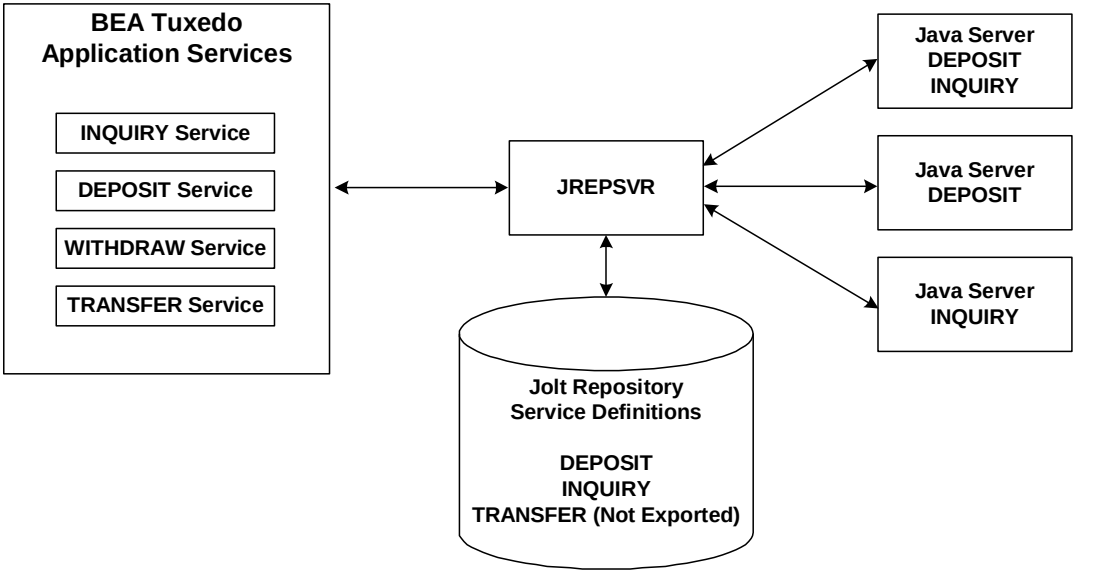
For each BEA Tuxedo service that you want to access using JET, an associated service definition *must* be stored in the Jolt Repository. In order to invoke a Tuxedo service, its service definition must also be explicitly *exported*, or made available, to Java servers. All Repository services that are exported to one Java server are exported to all Java servers. BEA Tuxedo handles the cases where subsets of services may be needed for one client and not others. You specify and export service definitions using the graphical and command-line tools described in "Tools for Managing Service Definitions" on page 1-9.

Before you can use JET, you must configure the JREPSVR in the `UBBCONFIG` file, as described in "Configuring JET for Java Server Access" on page 2-2. For each Jolt Repository, you can configure one or more JREPSVRs. Only one instance of the JREPSVR can be configured with read-write access to the Jolt Repository; all others are configured with read-only access.

How BEA Tuxedo Services are Distributed

Figure 1-2 illustrates how the JREPSVR distributes BEA Tuxedo services to multiple Java servers. In this example, the BEA Tuxedo server has four services, but only three are defined in the Jolt Repository. The WITHDRAW service is not defined and therefore is unavailable to Java servers. In addition, the TRANSFER service is defined in the Jolt Repository but is not exported, or made available, to the Java servers. The Java servers are able to invoke only the DEPOSIT and INQUIRY services.

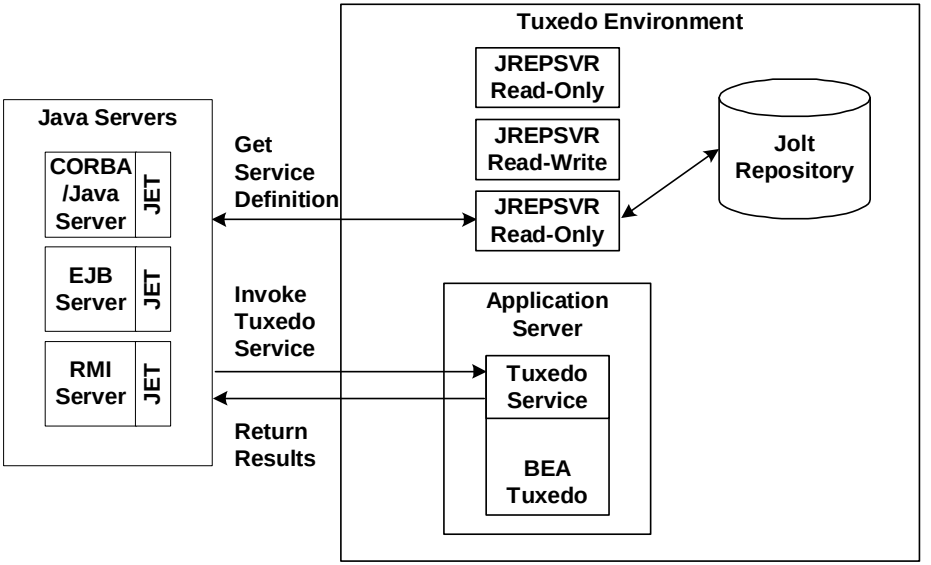
Figure 1-2 Distributing BEA Tuxedo Services via JET



Workflow for Handling BEA Tuxedo Service Requests

Figure 1-3 shows how JREPSVRs handle BEA Tuxedo service requests from Java servers.

Figure 1-3 Workflow for Handling BEA Tuxedo Service Requests



The workflow occurs in the following sequence:

1. The Java server submits a request for a BEA Tuxedo service.
2. JET retrieves the service definition for the requested service from the Jolt Repository using a JREPSVR. Because it is a read-only request, any available JREPSVR can service the request.
3. JET uses the service definition to translate buffer data. It submits the request to the BEA Tuxedo server and returns the results to the calling Java server.

Tools for Managing Service Definitions

This topic describes the following JET components, which are used for managing service definitions:

- Bulk Loader
- Jolt Repository Editor
- Jolt Servers
- Jolt Internet Relay

JET leverages these components from BEA Jolt for the sole purpose of managing service definitions used by the JREPSVR. In addition, the topic "Using BEA Tuxedo Buffer Types with JET" on page 1-13 describes the BEA Tuxedo buffer types that JET supports.

In order to use the Bulk Loader or Jolt Repository Editor to edit service definitions, you must first configure the Jolt servers. If you want to use these tools across a firewall, you must also configure Jolt Internet Relay. For configuration instructions, see Chapter 3, "Configuring JET for Client Access."

Bulk Loader

The Bulk Loader program populates service definitions for the JREPSVR. It uses service definitions that are specified in a specially-formatted text file. For more information about the Bulk Loader program, see Chapter 4, "Using the Bulk Loader Program."

Jolt Repository Editor

The Jolt Repository Editor is a GUI-based administration tool that allows developers and administrators to add, edit, delete, export, and test individual service definitions. You can modify parameters for BEA Tuxedo services, logically group BEA Tuxedo services into packages, and remove services from created packages. For more information, see Chapter 5, "Using the BEA Jolt Repository Editor."

Note: The Jolt Repository Editor controls services for JET client applications only. You cannot use it to make changes to the BEA Tuxedo application.

Jolt Servers

JET includes the following Jolt servers to handle communications with the JREPSVR when the Repository Editor or Bulk Loader programs are used:

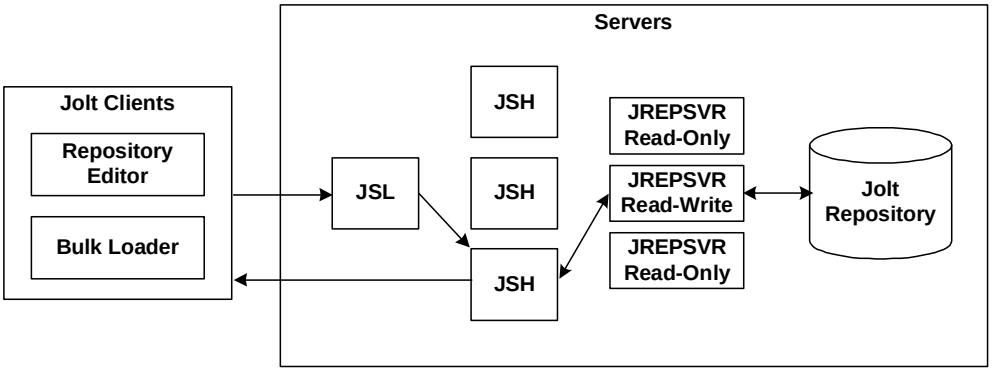
- **Jolt Server Listener (JSL)—**receives requests from clients and assigns them to an available Jolt Server Handler. The JSL is a BEA Tuxedo server.
- **Jolt Server Handler (JSH)**—manages network connectivity, executes service requests, and handles the translation of buffer data between BEA Tuxedo and Jolt buffers. Each JSH retains a cache copy of the Jolt Repository for performance purposes.

These servers work together to handle incoming requests from the Bulk Loader program or Jolt Repository Editor to update service definitions. The JSH communicates with the JREPSVR.

Note: You must configure these Jolt servers before you can update service definitions. For more information, see Chapter 3, "Configuring JET for Client Access."

Figure 1-4 shows how the Jolt servers work together to handle requests from the Jolt Repository Editor or the Bulk Loader program.

Figure 1-4 Workflow for Handling Requests from the Jolt Clients



The workflow occurs in the following sequence:

1. The Jolt Repository Editor or Bulk Loader sends an access request to the JSL.
2. The JSL forwards the request to an available JSH.

3. The JSH directs the request to an available JREPSVR. If the request is for write-access to the Jolt Repository, the JSH directs the request to the one JREPSRVR that has write-access to the Jolt Repository.
4. The JREPSVR processes the request in the Jolt Repository and returns the results to the JSH.
5. The JSH saves the results in its cache and returns the results to the client application that made the request.

   The JSL also updates the Jolt Repository cache in other JSHs.

Jolt Internet Relay

You use Jolt Internet Relay only if you want to run the Jolt Repository Editor or Bulk Loader and access Tuxedo service definitions on the other side of a firewall. Jolt Internet Relay handles message routing from the Jolt Repository Editor or Bulk Loader running *outside* a firewall to a JSL or JSH *behind* a firewall.

Components of Jolt Internet Relay

Jolt Internet Relay consists of the following components:

- **Jolt Relay** (**JRLY**)**—**a standalone software component that routes Jolt messages to the Jolt Relay Adapter. Requiring only minimal configuration to work with Jolt clients, the Jolt Relay eliminates the need for the BEA Tuxedo system to run on the same machine as the Web server.
- **Jolt Relay Adapter** (**JRAD**)**—**a BEA Tuxedo system server, but does not include any BEA Tuxedo services. It requires command-line arguments to allow it to work with the JSH and the BEA Tuxedo system. JRAD receives client requests from JRLY, and forwards the request to the appropriate JSH. Replies from the JSH are forwarded back to the JRAD, which sends the response back to the JRLY.

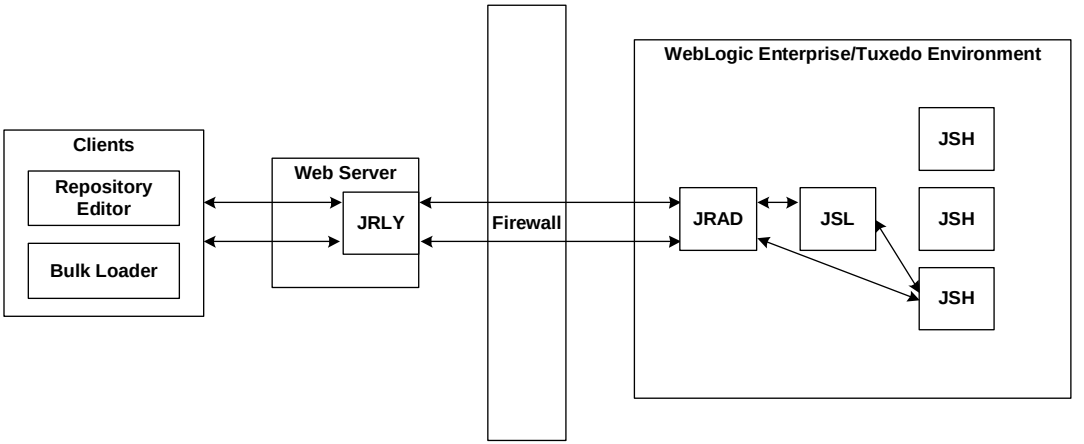
A single Jolt Internet Relay (JRLY/JRAD pair) handles multiple clients concurrently.

Note: You must configure Jolt Internet Relay before you can use the Repository Editor or Bulk Loader outside a firewall. For more information, see Chapter 3, “Configuring JET for Client Access.”

Workflow of Client Requests Using Jolt Internet Relay

Figure 1-5 shows how the JRLY and JRAD work together to route requests from the Jolt Repository Editor or the Bulk Loader program to a JSH across a firewall.

Figure 1-5 Workflow for Jolt Internet Relay



The workflow occurs in the following sequence:

1. The Jolt Repository Editor or Bulk Loader sends an access request to the JRLY on the Web server.
2. The JRLY forwards the request across the firewall to the JRAD in the WebLogic Enterprise environment.
3. The JRAD forwards the message to the JSL.
4. The JSL forwards the request to an available JSH.
5. The JSH directs the request to an available JREPSVR (see Figure 1-4 for an example). If the request is for write-access to the Jolt Repository, the JSH directs the request to the one JREPSRVR that has write-access to the Jolt Repository.
6. The results are returned along the same route.
7. For subsequent access requests from the Jolt Repository Editor or Bulk Loader, the JRAD forwards the request to the selected JSH, bypassing the JSL.

Using BEA Tuxedo Buffer Types with JET

This topic describes the BEA Tuxedo buffer types that you use when defining parameters for a Tuxedo service call, as described in "Specifying Parameters for the BEA Tuxedo Service" on page 2-8. Using a buffer type involves the following steps:

1. Specifying the buffer type in the service definition using the Bulk Loader program or the Jolt Repository Editor.
2. Writing the code that uses the buffer specified in the service definition, as described in Chapter 2, "Invoking BEA Tuxedo Services."

Supported BEA Tuxedo Buffer Types

JET supports the following built-in BEA Tuxedo buffer types:

- FML, FML32
- VIEW, VIEW32
- X_COMMON
- X_C_TYPE
- CARRAY
- X_OCTET
- STRING

Note: X_OCTET is used identically to CARRAY.
X_COMMON and X_C_TYPE are used identically to VIEW.

For detailed information about the BEA Tuxedo typed buffers, data types, and buffer types, see the *BEA Tuxedo Programmer's Guide*.

Of the BEA Tuxedo built-in buffer types, the JET application programmer should be particularly aware of how JET handles the CARRAY (character array) and STRING built-in buffer types. The CARRAY type is used to handle data opaquely (for example, the characters of a CARRAY data type are not interpreted in any way). No data conversion is performed between a JET client and BEA Tuxedo service.

For example, if a BEA Tuxedo service uses a CARRAY buffer type and the user sets a 32-bit integer (in Java the integer is in big-endian byte order), then the data is sent unmodified to the BEA Tuxedo service. If the BEA Tuxedo service is run on a machine whose processor uses little-endian byte-ordering (for example, Intel processors), the BEA Tuxedo service must convert the data properly before the data can be used.

STRING Buffer Type

The STRING buffer type is a collection of characters. STRING consists of non-null characters and is terminated by a null character. The STRING data type is `character` and, unlike CARRAY, you can determine its transmission length by counting the number of characters in the buffer until reaching the null character.

Note: During the data conversion from JET to STRING, the null terminator is automatically appended to the end of the STRING buffers because a Java string is not null-terminated.

CARRAY Buffer Type

The CARRAY buffer type is a simple character array buffer type that is built into the BEA Tuxedo system. Because the system does not interpret the data (although the data type is known) when you use the CARRAY buffer type, you must specify a data length in the JET client application. The JET client must specify a data length when passing this buffer type.

To use the CARRAY buffer type, you first define the BEA Tuxedo service that you will be using with the buffer type. Then, write the code that uses the buffer type.

Note: X_OCTET is used identically to CARRAY.

FML Buffer Type

FML (Field Manipulation Language) is a flexible data structure that can be used as a typed buffer. The FML data structure stores tagged values that are typed, variable in length, and may have multiple occurrences. The typed buffer is treated as an abstract data type in FML.

FML gives you the ability to access and update data values without having to know how the data is structured and stored. In your application program, you simply access or update a field in the fielded buffer by referencing its identifier. To perform the operation, the FML run time determines the field location and data type.

FML is especially suited for use with JET clients because the client and server code can be in two languages (for example, Java and C); the client/server platforms can have different data type specifications; or the interface between the client and the server can change frequently.

VIEW Buffer Type

VIEW is a built-in BEA Tuxedo typed buffer. The VIEW buffer provides a way to use C structures and COBOL records with the BEA Tuxedo system. The VIEW typed buffer enables the BEA Tuxedo run-time system to understand the format of C structures and COBOL records based on the view description that is read at run time.

When allocating a VIEW, your application specifies a VIEW buffer type and a subtype that matches the name of the view (the name that appears in the view description file). The parameter name must match the field name in that view. Because the BEA Tuxedo run-time system can determine the space needed based on the structure size, your application need not provide a buffer length. The run-time system can also automatically handle such things as computing how much data to send in a request or response, and handle encoding and decoding when the message transfers between different machine types.

Comparison Between Jolt and JET

This topic includes the following sections:

- Overview of Jolt and JET
- Architecture Comparison
- Components Comparison
- Functionality Comparison
- Class Library Comparison

Overview of Jolt and JET

WebLogic Enterprise provides two technologies that enable Java applications to invoke BEA Tuxedo services:

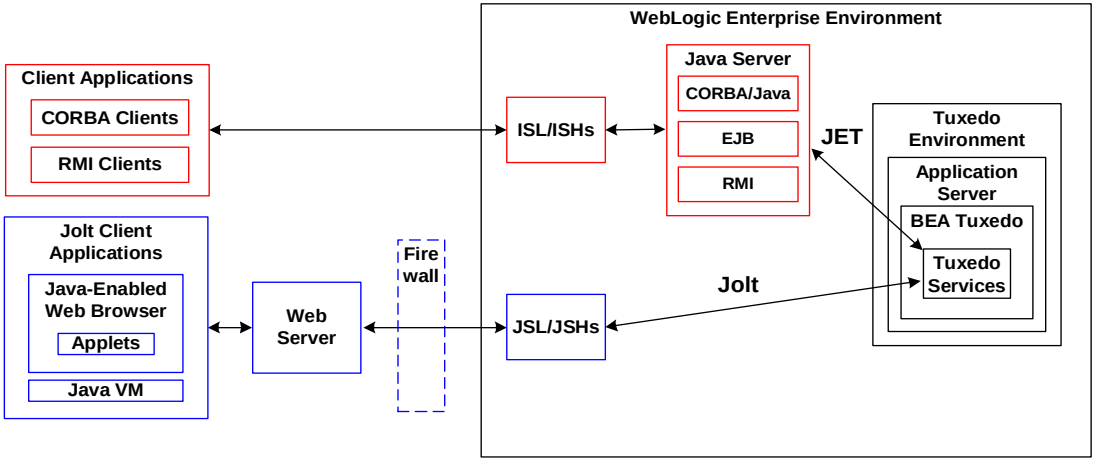
- **Jolt** provides *client-side* access to BEA Tuxedo services. Jolt enables browser-based clients (applications, such as the Jolt Bulk Loader program, or applets such as the Jolt Repository Editor) to invoke BEA Tuxedo services and process the results. Jolt is an optional WebLogic Enterprise component that is installed separately from WebLogic Enterprise.
- **Java Enterprise Tuxedo (JET)** provides *server-side* access to BEA Tuxedo services. JET enables Java servers (CORBA/Java, EJB, or RMI servers) running *within* the WebLogic Enterprise domain to invoke BEA Tuxedo services and process the results. JET is automatically installed when you install WebLogic Enterprise.

The technology that you use depends on the nature of your application. Use Jolt to provide client applets with access to BEA Tuxedo services, or use JET to provide Java servers, running in the WebLogic Enterprise domain, with access to BEA Tuxedo services.

Architecture Comparison

Figure 1-6 provides an overview of the Jolt and JET architectures.

Figure 1-6 Overview of Jolt and JET Architectures



At run time, requests for BEA Tuxedo services are handled differently:

- For servlets that make service requests using Jolt, requests and replies are routed to the BEA Tuxedo service through the Java Listener (JSL) to a Java Handler (JSH).
- For Java servers that make service requests using JET, requests and replies are routed directly through JET, without using the JSL/JSH. The server-side object lives inside JavaServer(.exe). Because the JavaServer(.exe) is a regular Tuxedo server, it can call Tuxedo services within its domain or across domains.

For more detailed information about Jolt, see the BEA Jolt documentation in the WebLogic Enterprise online documentation.

Components Comparison

Because JET runs in the WebLogic Enterprise environment, it does not require all of the components that Jolt uses. Table 1-2 compares the components used in Jolt and JET.

Table 1-2 Comparison of Jolt and JET Components

Component	Used in Jolt?	Used in JET?
Class Library	Yes	Yes
JoltBeans	Yes	No
Jolt Repository	Yes	Yes
Jolt Repository Editor	Yes	Yes
Bulk Loader	Yes	Yes
Jolt Repository Server (JREPSVR)	Yes	Yes
Jolt Servers: ■ Jolt Listener (JSL) ■ Jolt Handler (JSH)	Yes	Yes
Jolt Internet Relay: ■ Jolt Relay (JRLY) ■ Jolt Relay Adapter (JRAD)	Yes	Yes
Servlet Connectivity for Tuxedo	Yes	No
ASP Connectivity for Tuxedo	Yes	No

At run time, Java servers using JET require only the JET Class Library, the Jolt Repository, and the JREPSVR. All other components are used only when editing BEA Tuxedo service definitions in the Jolt Repository, as described in “Tools for Managing Service Definitions” on page 1-9.

Functionality Comparison

Table 1-3 compares the major features used in Jolt and JET.

Table 1-3 Comparison of Jolt and JET Functionality

Function	Jolt	JET
Calling code	Applets launched inside web browsers	Java servers: CORBA/Java, EJB, and RMI servers
Sessions	Yes	No
Transactions	Explicit; client must establish transactions explicitly.	Implicit; calls can be excluded from transactions.
Events	Yes	No
Security	56-bit and 128-bit	40-bit only
Class for invoking BEA Tuxedo service	`JoltRemoteService` class	`JoltService` class
Jolt version	Jolt 1.2.1	Jolt 1.2

Class Library Comparison

The Jolt and JET class libraries share some common components. This topic describes the packages, interfaces, and classes that Jolt and JET use.

Packages

Table 1-4 describes the `bea.jolt.*` packages that Jolt and JET use. The only common package is `bea.jolt`, of which JET uses the subset described in Table 1-5.

Table 1-4 Packages Used in Jolt and JET

Package Name / Component	Used in Jolt?	Used in JET?
`package bea.jolt`	Yes	Yes
`package bea.jolt.beans`	Yes	No
`package bea.jolt.beans.awt`	Yes	No
`package bea.jolt.beans.swing`	Yes	No
`package bea.jolt.beans.swi ng11`	Yes	No
`package bea.jolt.pool`	Yes	No
`package bea.jolt.pool.asp`	Yes	No
`package bea.jolt.pool.servlet`	Yes	No
`package bea.jolt.pool.servlet.weblogic`	Yes	No

Package bea.jolt Components

Table 1-5 describes the components of the `bea.jolt` package that Jolt and JET use. JET uses a subset of the `bea.jolt` package.

Table 1-5 Components of the bea.jolt Package

Package Name / Component	Used In Jolt?	Used in JET?
Interfaces		
`Message`	Yes	No
Classes		
`JoltAdmSession`	Yes	No
`JoltDefinition`	Yes	Yes
`JoltMessage`	Yes	No
`JoltParam`	Yes	No
`JoltRemoteService`	Yes	Yes
`JoltReply`	Yes	No
`JoltRepository`	Yes	No
`JoltRequest`	Yes	No
`JoltService`	No	Yes
`JoltServiceBase`	No	Yes
`JoltSession`	Yes	No
`JoltSessionAttributes`	Yes	No
`JoltTransaction`	Yes	No
`JoltUserEvent`	Yes	No
`LockMonitor`	Yes	No
`SBuffer`	Yes	Yes
`Session`	Yes	No
Exceptions		
`ApplicationException`	Yes	Yes

Table 1-5 Components of the bea.jolt Package (Continued)

Package Name / Component	Used In Jolt?	Used in JET?
`DefinitionException`	Yes	No
`EventException`	Yes	No
`JoltException`	Yes	Yes
`MessageException`	Yes	Yes
`ServiceException`	Yes	Yes
`SessionException`	Yes	No
`TransactionException`	Yes	No

Note: Jolt application developers must use the `JoltRemoteService` class, while JET application developers should use the `JoltService` class. Differences exist between the constructors and the `call` method.

For detailed information about these classes, see the BEA Jolt Javadoc and the `bea.jolt` package in the WebLogic Enterprise API Javadoc in the WebLogic Enterprise online documentation.

CHAPTER

2 Invoking BEA Tuxedo Services

This topic includes the following sections:

- Configuring JET for Java Server Access
- Invoking BEA Tuxedo Services with the JET Class Library

The JET Class Library provides developers with a set of object-oriented Java language classes for accessing BEA Tuxedo services. The `bea.jolt` package contains the JET Class Library. For more information about the classes that make up the JET Class Library, see the `bea.jolt` package in the WebLogic Enterprise API Javadoc in the WebLogic Enterprise online documentation.

Configuring JET for Java Server Access

This topic includes the following sections:

- Default Repository File
- Parameters to Specify in the UBBCONFIG File
- Sample UBBCONFIG File

Before a Java server application can use the JET Class Library to invoke BEA Tuxedo services, you must first define one or more Jolt Repository Servers (JREPSVRs) to manage service definitions stored in the Jolt Repository. The configuration must include a pointer to the location of the Jolt Repository file. You must configure these JREPSVRS on the same host on which the BEA Tuxedo services are running. For more information about the JREPSVR, see "Jolt Repository Server" on page 1-5.

To define JREPSVRs, you change settings in the `GROUPS` and `SERVERS` section of the `UBBCONFIG` file for the application. The `UBBCONFIG` file is an ASCII version of the BEA Tuxedo configuration file. For each `UBBCONFIG` file, you can configure a single Jolt Repository. Thereafter, you must create the `tuxconfig` file using `tmloadcf`, set environment variables (including `TUXDIR` and `CLASSPATH`), and then boot the Tuxedo application using `tmboot`, as described in "Starting and Shutting Down Applications" in the *Administration Guide*.

Default Repository File

JET includes a sample Jolt Repository file, `jrepository` (in the `udataobj\jolt` directory), which includes service definitions for the BEA Tuxedo TOUPPER service, along with other sample service definitions. Start with the `jrepository` file provided with the installation, even if you are not going to use the service definitions it contains. You can simply delete the packages or services that you do not need.

In addition, the CORBA/Java-to-Simpapp and EJB-to-Tuxedo Simpapp sample applications include local repository files that contain the service definitions needed to run the samples. For more information about these sample applications, see *CORBA, J2EE, and Tuxedo Interoperability and Coexistence*.

Warning: Do not modify the Repository files manually or you will not be able to use the Jolt Repository Editor. Although the `jrepository` file can be modified and read with any text editor, JET does not have integrity checks to ensure that the file is in the proper format. Any manual changes to the `jrepository` file might not be detected until run time. For more information, see "Using the BEA Jolt Repository Editor" on page 5-1.

Parameters to Specify in the UBBCONFIG File

Table 2-1 describes the parameters in the `UBBCONFIG` file to specify.

Table 2-1 Parameters To Specify in UBBCONFIG File

Option	Parameters
`GROUPS`	`LMID`, `GRPNO`
`SERVERS`	`SRVGRP`, `SRVID`

For more information about the `UBBCONFIG` file, see "Creating a Configuration File" in the *Administration Guide* and the *BEA Tuxedo Command Reference*.

GROUPS Section

A `GROUPS` entry is required for the group that includes the Jolt Repository. The group name parameter is a name selected by the application. In the `GROUPS` section, complete the following steps:

1. Specify the same identifiers given as the value of the `LMID` parameter in the `MACHINES` section.
2. Specify the value of the `GRPNO`, between 1 and 30000.

SERVERS Section

The JREPSVR contains services for accessing and editing the Repository. Multiple JREPSVR instances share Jolt Repository information through a shared file.

Note: To achieve the fastest performance, configure the Jolt Repository on a local drive of the machine on which the Java server runs. At a minimum, the Jolt Repository must reside on a volume that is network accessible to the JREPSVR.

To configure JREPSVR, modify the `SERVERS` section of the `UBBCONFIG` file by completing the following steps:

1. Indicate a new server identification (for example, `98`) with the `SRVID` parameter.
2. Specify the `-W` flag for one (and only one) JREPSVR to ensure that you can edit service definitions using the Bulk Loader program or the Jolt Repository Editor.

   Without explicitly setting this flag, the Repository is read-only.

   Note: You must install only one writable JREPSVR (that is, only one JREPSVR with the `-W` flag). Multiple read-only JREPSVRs can be installed on the same host.
3. Type the `-P` flag to specify the path of the Jolt Repository file.

   Note: An error message is displayed in the BEA Tuxedo `ULOG` file if the argument for the `-P` flag is not entered.
4. Add the file pathname of the Repository file (for example, `/app/jrepository`).
5. Boot the BEA Tuxedo system by using the `tmloadcf` (for example, `tmloadcf -y ubbconfig`) and `tmboot` commands.

   For information about `tmloadcf` and `tmboot`, see *Administering a BEA Tuxedo Application at Run Time*.

Sample UBBCONFIG File

Listing 2-1 shows relevant portions of a sample `UBBCONFIG` file.

Listing 2-1 Sample UBBCONFIG File

```
*GROUPS
   JREPGRP        GRPNO=94 LMID=SITE1
*SERVERS
   JREPSVR SRVGRP=JREPGRP SRVID=98
```

```
   RESTART=Y GRACE=0 CLOPT="-A -- -W -P /app/jrepository"
   JREPSVR SRVGRP=JREPGRP SRVID=97
   RESTART=Y RQADDR=JREPQ GRACE=0 CLOPT="-A -- -P /app/jrepository"
   JREPSVR SRVGRP=JREPGRP SRVID=96
   RESTART=Y RQADDR=JREPQ REPLYQ=Y GRACE=0 CLOPT="-A -- -P
/app/jrepository"
```

Notes: For the CLOPT parameter, the pathname of the file must match the argument of the `-P` option.

For UNIX systems, use the slash (`/`) when setting the path to the `jrepository` file (for example, `app/repository`). For Windows NT systems, use the backslash (`\`) and specify the drive name (for example, `c:\app\repository`).

Invoking BEA Tuxedo Services with the JET Class Library

This topic includes the following sections:

- Importing Packages
- Instantiating a JoltService Object
- Specifying Parameters for the BEA Tuxedo Service
- Calling the BEA Tuxedo Service
- Handling Results

This topic shows how to invoke a BEA Tuxedo service using the JET Class Library. It uses sample Java code from the EJB-to-Tuxedo Simpapp sample application, as well as some code fragments from unshipped applications to illustrate other programming techniques. The EJB-to-Tuxedo Simpapp sample application invokes a BEA Tuxedo service, TOUPPER, that converts a text string to all uppercase letters. The EJB-to-Tuxedo Simpapp sample application is located in `samples\interop\ejb_tux`.

This topic walks through the portions of the SimpBean code that use the JET Class Library. For a complete description of this sample application, see "EJB-to-Tuxedo Simpapp Sample Application" in *CORBA, J2EE, and Tuxedo Interoperability and Coexistence*.

Note: At run time, the BEA Tuxedo service(s) that you want to invoke must be running on the BEA Tuxedo server. The configuration settings specified in "Configuring JET for Java Server Access" must be activated for the Tuxedo application.

Importing Packages

To use JET, an application must import the `bea.jolt.*` package, which contains the JET Class Library. Listing 2-2, from the EJB-to-Tuxedo Simpapp sample application, shows how the SimpBean imports the required packages, including the `bea.jolt` package.

Listing 2-2 Importing the bea.jolt Package

```
package ejb;

import java.rmi.*;
import javax.ejb.*;

import bea.jolt.*;
```

Instantiating a JoltService Object

The `joltNativeCall` method is where the SimpBean invokes the TOUPPER BEA Tuxedo service using the JET Class Library. The key component of the Jolt Class Library is the `JoltService` class.

To access a BEA Tuxedo service using JET, an application begins by creating an instance of the `JoltService` class. The `JoltService` object represents the BEA Tuxedo service that you want to invoke. The constructor requires the name of the BEA Tuxedo service. JET uses this service name to locate the associated service definition in the Jolt Repository.

Listing 2-3, from the EJB-to-Tuxedo Simpapp sample application, shows the declaration of the `joltNativeCall` method and the instantiation of a `JoltService` to represent the BEA Tuxedo TOUPPER service.

Listing 2-3 Creating an Instance of the JoltService Class

```
  String joltNativeCall (String svcName, String data)
```

```
{
  JoltService svc;

  try {
    svc = new JoltService (svcName);
```

Specifying Parameters for the BEA Tuxedo Service

After instantiating a `JoltService`, an application specifies parameters for the BEA Tuxedo service. The JoltService class provides `addXXXX` methods that are used to specify the parameters. For each parameter, the application specifies the buffer type (such as `STRING`, `CARRAY`, `VIEW`, and `FML`) and the buffer value. For more information, see "Using BEA Tuxedo Buffer Types with JET" on page 1-13.

Note: The parameters must be valid for the BEA Tuxedo service. If invalid parameters are specified, the BEA Tuxedo service will return an error.

Specifying String Parameters

Listing 2-4, from the EJB-to-Tuxedo Simpapp sample application, shows the call to the `addString` method, passing the buffer type (`STRING`) and the text string (`data`) supplied by the user.

Listing 2-4 Specifying the Parameters for the TOUPPER Service

```
      svc.addString("STRING", data);
```

Specifying Array Parameters

Listing 2-5 is a code fragment that shows how to specify array elements for two array parameters: a String array named `svarray` and a short array named `harray`.

Listing 2-5 Specifying Array Parameters

```
j_service.addString("svarray", "String[0]");
j_service.addString("svarray", "String[1]");
j_service.addShort("harray", (short) (600));
j_service.addShort("harray", (short) (700));
```

Specifying Parameters of Various Data Types

Listing 2-6 is a code fragment that shows how to specify parameters of various data types.

Listing 2-6 Specifying Parameters of Various Data Types

```
j_service.addInt("ctime", 100);
j_service.addString("hname", "***NoName***");
j_service.addFloat("fval", (float) 200.00);
j_service.addByte("cval", (byte) 'S');
j_service.addInt("lval", 300);
j_service.addString("sval","10  bytes(unexpected string)");
j_service.addShort("hval", (short) 400);
j_service.addDouble("dval", 500.00);
```

Calling the BEA Tuxedo Service

After the parameters are specified, an application calls the `call` method, which submits a service request to BEA Tuxedo. The `call` method provides two syntax options:

- An application uses `call()` to have the BEA Tuxedo service call *participate* in the current transaction, if applicable. JET uses the implicit transaction mechanisms provided by WebLogic Enterprise and BEA Tuxedo. If the BEA Tuxedo service call is part of a transaction and the transaction is rolled back, any changes made by the BEA Tuxedo service call are rolled back automatically.
- An application uses `call(null)` to *exclude* the BEA Tuxedo service call from participating in the current transaction. Pass `null` only if you are certain that any

operations performed by the BEA Tuxedo service will never need to be rolled back.

Listing 2-7, from the EJB-to-Tuxedo Simpapp sample application, shows the call to the `call` method, passing `null` because the operation performed by the TOUPPER service is not transactional.

Listing 2-7 Invoking the BEA Tuxedo TOUPPER Service

```
svc.call (null);
```

At this point, JET obtains the TOUPPER service definition from the Jolt Repository, converts the buffer data to BEA Tuxedo buffers, submits the request to BEA Tuxedo, and awaits a reply.

Handling Results

This section describes how to handle the results (exceptions and parameters) returned from a Tuxedo service call.

Handling Exceptions

The JET Class Library returns JET and BEA Tuxedo errors as exceptions. For a complete list of JET exceptions, see the WebLogic Enterprise *Javadoc* in the WebLogic Enterprise online documentation.

Listing 2-8, from the EJB-to-Tuxedo Simpapp sample application, shows the code to catch a `ServiceException`.

Listing 2-8 Handling the Results of the BEA Tuxedo Service Call

```
} catch (ServiceException e) {
  System.out.println("JoltService got "+e);
  return new String("");
```

Handling Returned Parameters

This section describes how to handle parameters that were returned from the BEA Tuxedo service call. The `JoltService` class provides `getXXXX` methods that are used to retrieve individual parameters. The application passes the name of the parameter to retrieve, as well as a default value in case no value is returned.

Retrieving String Parameters

Listing 2-9, from the EJB-to-Tuxedo Simpapp sample application, shows how to retrieve the String parameter, specified in Listing 2-4, that was returned by the TOUPPER service. In the call to the `getStringDef` method, the application passes the parameter name (`STRING`) and the default value (`"no_response"`) if no value was returned.

Listing 2-9 Retrieving a String Parameter

```
    return svc.getStringDef("STRING", "no_response");
```

Retrieving Arrays

Listing 2-10 is a code fragment that shows how to handle the returned array elements for the two arrays, `svarray` and `harray`, that were specified in Listing 2-5.

Listing 2-10 Retrieving Returned Arrays

```
// retrieve svarray elements
for (int i = 0; i < 2; i++)
{
    String s;
    if ((s = j_service.getStringItemDef(
       "svarray", i, null)) == null)
     break;
System.out.println( hdr+"]: svarray["+i+"]="+s);
}
// retrieve harray elements
for (int i = 0; i < 2; i++)
   {
    short h;
```

```
    if ((h = j_service.getShortItemDef(
       "harray", i, (short) 0)) == 0)
     break;
    System.out.println( hdr+"]: harray["+i+"]="+h);
}
```

Retrieving Parameters of Various Data Types

Listing 2-11 is a code fragment that shows how to handle the returned parameters, of various data types, that were specified in Listing 2-6.

Listing 2-11 Retrieving Parameters of Various Data Types

```
System.out.println(hdr+"]: ctime="+
j_service.getIntDef("ctime", 0)+
nhdr+"]: hname="+
j_service.getStringDef("hname", null)+
nhdr+"]: sval="+
j_service.getStringDef("sval", null)+
nhdr+"]: fval="+
j_service.getFloatDef("fval",(float) 0.0)+
nhdr+"]: cval="+
j_service.getByteDef("cval", (byte) '.')+
nhdr+"]: lval="+
j_service.getIntDef("lval", 0)+
nhdr+"]: hval="+
j_service.getShortDef("hval", (short) 0)+
nhdr+"]: dval="+
j_service.getDoubleDef("dval",(double)0.0));
```

3 Configuring JET for Client Access

This topic includes the following sections:

- Configuring JET
- JET Administrative Reference

The section "Configuring JET" on page 3-2 provides the instructions you need to configure JET for client access. You need to complete these instructions *only if* you need to add or edit BEA Tuxedo service definitions using either of the following client programs: the Bulk Loader program or the Jolt Repository Editor. The section "JET Administrative Reference" on page 3-8 provides supplemental reference information.

Note: Throughout this topic, the term *client* refers to either the Bulk Loader or the Jolt Repository Editor.

This topic assumes that you are familiar with BEA Tuxedo and that you have experience with the operating systems and network environment in which you are configuring JET. It also assumes that you have already configured the Jolt Repository Server (JREPSVR) according to the configuration instructions in "Configuring JET for Java Server Access" on page 2-2.

Configuring JET

This topic provides instructions for configuring JET for client access. It includes the following sections:

- About Configuring JET for Client Access
- Step 1: Configure JSL
- Step 2: Configure Jolt Relay
- Step 3: Registering Tuxedo Services with the Repository

About Configuring JET for Client Access

Before you can use the Bulk Loader program or Jolt Repository Editor to add or edit BEA Tuxedo service definitions, you must configure and start the following servers:

- **Jolt Listener (JSL)—**receives requests from clients and assigns them to an available Jolt Handler (JSH). A JSH manages network connectivity, executes service requests, and handles the translation of buffer data between BEA Tuxedo and Jolt buffers.
- **Jolt Relay (JRLY)**—handles communications *across* a firewall between clients and the JSL. You do *not* need to configure Jolt Relay if no firewall exists between the client and JSL.

For an introduction to these components, see “Tools for Managing Service Definitions” on page 1-9.

Note: Before you proceed with this topic, you first need to configure the Jolt Repository Server (JREPSVR) according to the configuration instructions in “Configuring JET for Java Server Access” on page 2-2.

For more information about the `UBBCONFIG` file, see “Creating a Configuration File” in the *Administration Guide*.

Step 1: Configure JSL

To configure JET for client access, you must configure the JSL on the host on which the BEA Tuxedo services that you want to invoke are running. You configure the JSL in the UBBCONFIG file. For an introduction to the JSL, see "Jolt Servers" on page 1-10. For additional information about configuration parameters, see "Jolt Server Reference" on page 3-8.

Note: Before you begin, be sure to set the CLASSPATH to include the directory in which the jolt.jar file resides (such as udataobj\jolt).

To configure the JSL:

1. Open the UBBCONFIG file with a text editor.
2. In the MACHINES section, specify MAXWSCLIENTS=*number* (Required).

Note: If MAXWSCLIENTS is not set, JSL does not boot.

3. In the GROUPS section, set GROUPNAME *required parameters* [*optional parameters*].
4. Set the SERVERS section (Required).

   Lines within this section have the form:

   JSL *required parameters* [*optional parameters*]

   where JSL specifies the filename (*string_value*) of the JSL to be executed by tmboot(1), as described in the following step.

5. Set the following required parameters for JSL:

   SVRGRP=*string_value*

   SRVID=*number*

   CLOPT="-A...-n...//*host port*"

   For more information about these parameters, see "Creating a Configuration File" in the *Administration Guide*.

6. Set the following optional parameters for JSL, if you want:

   MIN # of JSHs

   MAX # of JSHs

Upon startup, the JSL starts the configured minimum number of JSHs. As more concurrent requests are received, it might start additional JSHs to handle the request load, up to the configured maximum number of JSHs.

To use these parameters, you first need to understand how doing so affects your application. For more information about these parameters, see "Creating a Configuration File" in the *Administration Guide*.

Step 2: Configure Jolt Relay

To configure JET for client access *across* a firewall, you must also configure Jolt Internet Relay, which includes the following components:

- **Jolt Relay** (**JRLY**)**—**routes Jolt messages to the Jolt Relay Adapter. JRLY runs on a Web server *outside* the firewall.
- **Jolt Relay Adapter** (**JRAD**)**—**receives client requests from JRLY and forwards the request to the appropriate JSH. The JSH forwards replies back to the JRAD, which sends the response back to the JRLY. JRAD runs as a Tuxedo server in the Tuxedo environment *behind* the firewall. The JRAD does not need to be in the same `APPDIR` as the JSL/JSH servers.

Note: You do not need to configure Jolt Relay if no firewall exists between client programs and the JSL.

For an introduction to Jolt Internet Relay, see "Jolt Internet Relay" on page 1-11. For additional information about configuration parameters, see "Jolt Internet Relay Reference" on page 3-17.

Configuring JRLY on the Web Server

To configure JRLY, you first start JRLY on the Web server and then change the configuration file. Be sure to follow the instructions for your Web server platform—the instructions for UNIX and Windows NT are slightly different. For a detailed description of JRLY configuration parameters, see "Jolt Internet Relay Reference" on page 3-17.

Note: The format for directory and filenames is determined by the operating system. UNIX systems use the forward slash (/). Windows NT systems use the backslash (\). If any files specified in `LOGDIR`, `ACCESS_LOG`, or `ERROR_LOG` cannot be opened for writing, JRLY prints an error message on `stderr` and exits.

Installing and Starting JRLY (Windows NT Only)

JRLY runs as an NT service in the Windows NT environment. To install JRLY on the Web server machine, complete the following steps:

1. Create a directory for Jolt Relay on the Web server machine.
2. Copy the contents of the `/udataobj/jolt/relay` directory to the directory you created on the Web server machine.
3. Install JRLY as an NT service by typing the following command at the system prompt:

   ```
   jrly -install
   ```

   By default, the JRLY service is configured to start automatically.
4. Update the registry with the full path of a new configuration file using the `jrly -set -f` command, as shown in the following example:

   ```
   jrly -set -f c:\tux71\udataobj\jolt\jrly.config
   ```

   In this example, the default JRLY Windows NT service (Jolt Relay) is assigned a configuration file called `jrly.config` that is located in the following directory: `c:\tux71\udataobj\jolt`.
5. Configure the service as needed using the Services Control Panel.

Starting JRLY (UNIX Only)

Start the JRLY process on UNIX by typing the following command at the system prompt:

```
jrly -f config_file
```

where *`config_file`* is the path and name of the JRLY configuration file. The default filename is `jrly.config`.

Note: If the specified configuration file does not exist or it cannot be opened, the JRLY writes a message to `stderr`, attempts to log the startup failure in the error log, and then exits.

Configuring JRLY (UNIX and Windows NT)

The configuration file uses a `TAG=VALUE` syntax. Blank lines or lines starting with the # character are ignored. Listing 3-1 shows an example of the formal specifications of the configuration file.

Listing 3-1 Formal Configuration File Specifications

```
LOGDIR=<LOG_DIRECTORY_PATH>
ACCESS_LOG=<ACCESS_FILE_NAME in LOGDIR>
ERROR_LOG=<ERROR_FILE_NAME in LOGDIR>
LISTEN=<IP:Port combination where JRLY will accept
comma-separated connections>
CONNECT=<IP:Port1, IP:Port2...IP:PortN:Port(List of IP:Port
combinations associated with JRADs: can be 1...N)>
```

Configuring the Socket Timeout (Windows NT only; optional)

The `SOCKETTIMEOUT` setting is specified in the JRLY configuration file. `SOCKETTIMEOUT` is the time, in seconds, for which JRLY Windows NT service blocks for network activity (new connections, data to be read, and closed connections). `SOCKETTIMEOUT` also affects the Service Control Manager (SCM). When the SCM requests the Windows NT service to stop, the SCM must wait for at least `SOCKETTIMEOUT` seconds before quitting.

Table 3-1 describes the formats for the host names and the port numbers.

Table 3-1 Host Name and Port Number Formats

Host Name/Port #	Description
`//Hostname:Port`	`Hostname` is a string; `Port` is a decimal number.
`IP:Port`	`IP` is a dotted notation IP address; `Port` is a decimal number.

Configuring JRAD in the Tuxedo Environment

To configure JRAD, a Tuxedo server, you first start JRAD in the BEA Tuxedo environment and then change the configuration file.

Starting the Jolt Relay Adapter (JRAD)

To start the Jolt Relay Adapter, complete the following steps:

1. Type `tmloadcf -y ubbFile`, where *`ubbFile`* is the name of the `UBBCONFIG` file associated with this JRAD.
2. Type `tmboot` to boot the JRAD server.

Configuring the JRAD

While configuring the JRAD, consider the following rules:

- A single JRAD process can be connected to only one JRLY.
- A JRAD can be configured to communicate with only one JSL and its associated JSHs.
- Multiple JRADs can be configured to communicate with one JSL.
- The `CLOPT` parameter for BEA Tuxedo services must be included in the `UBBCONFIG` file.

To configure the JRAD, complete the following steps:

1. Type `-l` *`hexadecimal format`*. (The JSL port to which the JRLY connects on behalf of the client.)
2. Type `-c` *`hexadecimal format`*. (The address of the corresponding JSL to which JRAD connects.)

   Note: The format is 0x0002PPPNNN or, in dot notation, 100.100.10.100.
3. Configure networked components.

Step 3: Registering Tuxedo Services with the Repository

In order to make the JET services available to Java servers, you must define the BEA Tuxedo services that use BEA Tuxedo. To define the Tuxedo service:

1. Build the BEA Tuxedo server that contains the service. For more information, see the *BEA Tuxedo Application Development Guide*.
2. For each Tuxedo service that you want to invoke, you must register its service definition in the Jolt Repository.
   - To populate the Jolt Repository with service definitions defined in a bulk loader file, see Chapter 4, "Using the Bulk Loader Program."
   - To add or edit service definitions with the Jolt Repository Editor, see Chapter 5, "Using the BEA Jolt Repository Editor."

   Note: You cannot use the Bulk Loader or Jolt Repository Editor *until* the JREPSVR and JSL are properly configured and running. In addition, if a firewall exists between the Jolt Repository Editor and the JSL, Jolt Internet Relay must also be properly configured and running.

JET Administrative Reference

This topic includes detailed supplemental reference information for the following JET components:

- Jolt Server Reference
- Jolt Internet Relay Reference

Jolt Server Reference

This section provides supplemental reference information for the Jolt Listener (JSL) and Jolt Handler (JSH). For an introduction to these servers, see "Jolt Servers" on page 1-10. For configuration instructions, see "Step 1: Configure JSL" on page 3-3.

About Jolt Servers

JET provides the following Jolt servers:

- **Jolt Server Listener (JSL)—**is configured to support clients on an IP/port combination.The JSL works with one or more Jolt Server Handlers (JSHs) to provide client connectivity to the backend of the WebLogic Enterprise system. The JSL runs as a BEA Tuxedo server.
- **Jolt Server Handler (JSH)—**is a program that runs on a BEA Tuxedo server machine to provide a network connection point for remote clients. The JSH works with the JSL to provide client connectivity residing on the backend of the WebLogic Enterprise system. One ore more JSHs can be available to the JSL (up to 32767). For additional information, see the description of the `-M` command-line option in "JSL Command-Line Options" on page 3-11.

System Administrator Responsibilities

The system administrator's responsibilities for the Jolt servers include:

- Determining the JSL network address.
- Determining the number of JET clients to be serviced. The number of clients to be serviced is limited by `MAXWSCLIENTS` setting in the `UBBCONFIG` file.
- Determining the minimum and maximum number of JSHs.

Starting the JSL

After you have configured the JSL in the `UBBCONFIG` file, you need to complete the following steps on the Tuxedo server to start all administrative and server processes in the `UBBCONFIG` **file:**

1. Type `tmloadcf`.

   This command parses the configuration file and loads the binary version of the configuration file.

2. Type `tmboot -y`.

   This command activates the application specified in the configuration file.

   If you do not enter any options, a prompt asks you if you really want to overwrite your `TUXCONFIG` file.

See *Administering a BEA Tuxedo Application at Run Time* or the *BEA Tuxedo Command Reference* for information about `tmloadcf` and `tmboot`.

Shutting Down the JSL

All shutdown requests to the Jolt servers are initiated by the BEA Tuxedo command:

```
tmshutdown -y
```

During shutdown:

- No new client connections are accepted.
- All current client connections are terminated. BEA Tuxedo rolls back open transactions. Each client receives an error message indicating that the service is unavailable.

Restarting the JSL

BEA Tuxedo monitors the JSL and restarts it in the event of a failure. When BEA Tuxedo restarts the listener process, the following events occur:

- Clients attempting a listener connection must try to reconnect. Clients attempting a handler connection receive a timeout or a time delay.
- Clients currently connected to a handler are disconnected (JSH exits when its corresponding JSL exits normally).

Configuring the JSL

The Jolt Server Listener (JSL) is a BEA Tuxedo server that is responsible for distributing connection requests from JET to an available Jolt Server Handler (JSH). BEA Tuxedo must be running on the host machine where the JSL and the JREPSVR are located.

JSL Command-Line Options

The server may need to obtain information from the command-line. The `CLOPT` parameter allows you to specify command-line options that can change some defaults in the server. Table 3-2 describes the JSL command-line options.

Table 3-2 JSL Command-Line Options

Option	Description
[-c *compression_threshold*]	Enables application data sent between a JET client and a Jolt server (JSH) to be compressed during transmission over the network. *compression_threshold* is a number that you specify between 0 and 2,147,483,647 bytes. Any messages that are larger than the specified compression threshold are compressed before transmission. The default is no compression; that is, if no compression threshold is specified, messages are not compressed on client or server. The previous -c *connection-mode* option has been replaced with the -j *connection-mode* option.
[-d *device_name*]	The device for platforms using the Transport Layer Interface. There is no default. Required. (Optional for sockets)
[-H *external netaddr*]	*external netaddr* is the network address JET that clients use to connect to the application. The JSL process uses this address to listen for clients attempting to connect at this address. If the address is 0x0002MMMMdddddddd and JSH network address is 0x00021111ffffffff, the known network address is 0x00021111dddd dddd. If the address starts with // network address, the type is IP based and the TCP/IP port number of JSH network address is copied into the address to form the combined network address. The IP address must be specified in the following format: -H //external ip address:MMMM (Optional for JSL in BEA Tuxedo 6.4 and 6.5)
[-I *init-timeout*]	The time (in seconds) that a JET client is allowed to complete initialization through the JSH before it is timed out by the JSL. Default is 60 seconds. (Optional)

Table 3-2 JSL Command-Line Options (Continued)

Option	Description
`[-j connection_mode]`	The following connection modes from clients are allowed: ■ RETAINED—the network connection is retained for the full duration of a session. ■ RECONNECT—the client establishes and brings down a connection when an idle timeout is reached, reconnecting for multiple requests within a session. ■ ANY—the server allows a client to request either a RETAINED or RECONNECT type of connection for a session. The default is ANY. That is, if no option is specified, the server allows a client to request either a RETAINED or RECONNECT type of connection. (Optional) **Note:** This option has been changed in this release from `-c [connection_mode]` to `-j [connection_mode]`.
`[-m minh]`	The minimum number of JSHs that are available in conjunction with the JSL at one time. The range of this parameter is from 0 through 255. Default is 0. (Optional)
`[-M maxh]`	The maximum number of JSHs that are available in conjunction with the JSL at one time. If this option is not specified, the parameter defaults to `MAXWSCLIENTS` divided by the rounded-up `-x` multiplexing factor (MPX). If specified, the `-M` option takes a value from 1 to 32767. (Optional)

Table 3-2 JSL Command-Line Options (Continued)

Option	Description
[-n *netaddr*]	Network address used by the Jolt listener with BEA Tuxedo 6.4 and 6.5, and WebLogic Enterprise 4.2, 5.0, and 5.1. TCP/IP addresses may be specified in the following formats: "*//host.name:port_number*" "*//#.#.#.#:port_number*" In the first format, the domain finds an address for *hostname* by using the local name resolution facilities (usually DNS). *hostname* must be the local machine, and the local name resolution facilities must unambiguously resolve hostname to the address of the local machine. This command-line option indicates the Jolt Server Handler. Default is JSH. (Optional) In the second example, the #.#.#.# is in dotted decimal format. In dotted decimal format, each # should be a number from 0 to 255. This dotted decimal number represents the IP address of the local machine. In both of the above formats, *port_number* is the TCP port number at which the domain process listens for incoming requests. *port_number* can either be a number between 0 and 65535 or a name. If *port_number* is a name, then it must be found in the network services database on your local machine. The address can also be specified in hexadecimal format when preceded by the characters "0x". Each character after the initial "0x" is a number from 0 to 9 or a letter from A to F (case insensitive). The hexadecimal format is useful for arbitrary binary network addresses such as IPX/SPX or TCP/IP. There is no default. (Required)
[-T *Client-timeout*]	The time (in minutes) allowed for a client to stay idle. If a client does not make any requests during this time, the JSH disconnects the client and the session is terminated. If an argument is not supplied, the session does not timeout. When the -j ANY or -j RECONNECT option is used, always specify -T with an idle timeout value. If -T is not specified and the connection is suspended, JSH does not automatically terminate the session. The session never terminates if a client abnormally ends the session. If a parameter is not specified, the default is no timeout. (Optional)
[-w *JSH*]	This command-line option indicates the Jolt Server Handler. Default is JSH. (Optional)

Table 3-2 JSL Command-Line Options (Continued)

Option	Description
[-x *mpx-factor*]	The number of clients that one JSH can service. Use this parameter to control the degree of multiplexing within each JSH process. If specified, this parameter takes a value from 1 to 32767 for UNIX and Windows NT. Default value is 10. (Optional)
[-Z 0|40|128]	When a network link between a JET client and the JSH is being established, this option allows encryption up to the specified level.The initial 0 means no DH nodes, no RC4. The numbers 56 and 128 specify the length (in bits) of the encryption key. The DH key exchange is needed to generate keys. Session keys are not transmitted over the network. The default value is 0.

Sample UBBCONFIG Settings for JSL

Listing 3-2 shows relevant portions of the UBBCONFIG file configured for JSL.

Listing 3-2 Sections of UBBCONFIG File Related to JSL Configuration

```
*MACHINES
   MACH1 LMID=SITE1
       MAXWSCLIENTS=40
*GROUPS
   JSLGRP     GRPNO=95   LMID=SITE1
*SERVERS
   JSL SRVGRP=JSLGRP SRVID=30 CLOPT= " -- -n 0x0002PPPPNNNNNNNN -d
/dev/tcp -m2 -M4 -x10"
```

The parameters shown in Table 3-3 are the only parameters that must be designated for the Jolt server groups and Jolt servers. You are not required to specify any other parameters.

Table 3-3 UBBCONFIG File Sections

Section	Parameters to Specify
MACHINES	MAXWSCLIENTS

Table 3-3 UBBCONFIG File Sections (Continued)

Section	Parameters to Specify
GROUPS	GRPNO, LMID
SERVERS	SRVGRP, SRVID, CLOPT

For more information about these parameters, see "Creating a Configuration File" in the *Administration Guide*.

MACHINES Section

The MACHINES section must contain an entry for each physical processor used by the application. Entries have the form:

ADDRESS or *NAME required parameters [optional parameters]*

where *ADDRESS* is the physical name of the processor, for example, the value produced by the UNIX system `uname -n` command.

LMID=*string_value*

This parameter specifies that the *string_value* is to be used in other sections as the symbolic name for ADDRESS. This name cannot contain a comma, and must be 30 characters or less. This parameter is required. There must be an LMID line for every machine used in a configuration.

MAXWSCLIENTS=*number*

The MAXWSCLIENTS parameter is required in the MACHINES section of the configuration file. It specifies the number of accessor entries on this processor to be reserved for Jolt and Workstation clients only. The value of this parameter must be between 0 and 32768, inclusive.

The Jolt server and Workstation use MAXWSCLIENTS in the same way. For example, if 200 slots are configured for MAXWSCLIENTS, this number configures BEA Tuxedo for the total number of remote clients used by Jolt and Workstation.

Note: Be sure to specify MAXWSCLIENTS in the configuration file. If it is not specified, the default is 0.

Note: If MAXWSCLIENTS is not set, the JSL does not boot.

GROUPS Section

A GROUPS entry is required for the group that includes the Jolt Server Listener (JSL). Make the GROUPS entry as follows:

1. The group name is selected by the application, for example: JSLGRP and JREPGRP.
2. Specify the same identifiers given as the value of the LMID parameter in the MACHINES section.
3. Specify the value of the GRPNO between 1 and 30000 in the *GROUPS section.

Note: Make sure that Resource Managers are *not* assigned as a default value for all groups in the GROUPS section of your UBBCONFIG file. Making Resource Managers the default value assigns a Resource Manager to the JSL and you receive an error during tmboot. In the SERVERS section, default values for RESTART, MAXGEN, and so on, are acceptable defaults for the JSL.

Lines within the GROUPS section have the form:

GROUPNAME *required parameters* [*optional parameters*]

where GROUPNAME specifies the logical name (*string_value*) of the group. The group name must be unique within all group names in the GROUPS section and LMID values in the MACHINES section. The group name cannot contain an asterisk(*), comma, or colon, and must be 30 characters or less.

SERVERS Section

Clients connect to Jolt Repository through the Jolt Server Listener (JSL). Services are accessed through the Jolt Server Handler (JSH). The JSL supports multiple clients and acts as a single point of contact for all the clients to connect to the application at the network address that is specified on the JSL command-line. The JSL schedules work for handler processes. A handler process acts as a substitute for clients on remote workstations within the administrative domain of the application. The handler uses a multiplexing scheme to support multiple clients on one port concurrently.

The network address specified for the JSL designates a TCP/IP address for both the JSL and any JSH processes associated with that JSL. The port number identified by the network address specifies the port number on which the JSL accepts new client connections. Each JSH associated with the JSL uses consecutive port numbers at the same TCP/IP address. For example, if the initial JSL port number is 8000 and there are a maximum of three JSH processes, the JSH processes use ports 8001, 8002, and 8003.

Note: Be sure to provide sufficient space between JSL port numbers (for example, use 8000, 8020, 8040, etc. for JSL port numbers). Misconfiguration of subsequent JSL port numbers results in a port number collision.

Security and Encryption

Authentication and key exchange data are transmitted between JET clients and the JSL/JSH using the DES key exchange and a 128-bit key, with 40 bits encrypted and 88 bits exposed.

Jolt Internet Relay Reference

This section provides supplemental reference information for the Jolt Relay (JRLY) and its associated Jolt Relay Adapter (JRAD). For an introduction to these components, see "Jolt Internet Relay" on page 1-11. For configuration instructions, see "Step 2: Configure Jolt Relay" on page 3-4.

About Jolt Relay and the Jolt Relay Adapter

The combination of the Jolt Relay (JRLY) and its associated Jolt Relay Adapter (JRAD) is typically referred to as the Internet Relay. Jolt Relay routes messages from a JET client to a JSL or JSH. This eliminates the need for the JSH and BEA Tuxedo to run on the same machine as the Web server (which is generally considered insecure). The Jolt Relay consists of the following two components:

- **Jolt Relay (JRLY)**—is the Jolt Relay frontend. It is not a BEA Tuxedo client or server and is not dependent on the BEA Tuxedo version. It is a standalone software component. It requires only minimal configuration to allow it to work with JET clients.
- **Jolt Relay Adapter (JRAD)**—is the Jolt Relay backend. It is a BEA Tuxedo system server, but does not include any BEA Tuxedo services. It requires command-line arguments to allow it to work with the JSL and the BEA Tuxedo system.

Note: The Jolt Relay is transparent to Java clients and servers. A Jolt server can simultaneously connect to Intranet clients directly, or through the Jolt Relay to Internet clients.

Jolt Relay

This section describes configuration options for JRLY.

Jolt Relay Failover

There are two points of failovers associated with JRLY:

- Jolt client to JRLY connection failover
- JRLY to JRAD adapter connection failover

Jolt Client to JRLY Connection Failover

If one server address does not result in a successful session, the failover function allows the Jolt Client API to connect to the next free (unconnected) JRLY specified in the argument list of the API. To enable this failover in the Windows NT environment, multiple Windows NT JRLY services can be executed. In a non-NT environment, multiple JRLY processes are executed. Each JRLY (service or process) has its own configuration file. This type of failover is handled by client API features that allow you to specify a list of Jolt server addresses (JSL or JRLY).

JRLY to JRAD Adapter Connection Failover

Each JRLY configuration file has a list of JRAD addresses. When a JRAD is unavailable, JRLY tries to connect to the next free (unconnected) JRAD, in a round-robin fashion. Two JRLYs cannot connect to the same JRAD. However, you can make the connection efficient by giving different JRAD address orders—if you make one extra JRAD available on standby, the first JRLY that loses its JRAD connects to the extra JRAD. This type of failover is handled by JRLY alone.

If any of the listed JRADs are not executing when JRLY is started, the initial connection fails. When a JET client tries to connect to JRLY, the JRLY again tries to connect to the JRAD.

To accommodate the failover functionality, you need to boot multiple JRADs by configuring them in the `UBBCONFIG` file.

Jolt Relay Process

The JRLY (frontend relay) process can be started before or after the JRAD is started. If the JRAD is not available when the JRLY is started, the JRLY attempts to connect to the JRAD when it receives a client request. If JRLY is still unable to connect to the JRAD, the client is denied access and a warning is written to the JRLY error log file.

Starting the JRLY on UNIX

Start the JRLY process by typing the command name at a system prompt:

```
jrly -f config_file
```

where *config_file* is the path and name of the JRLY configuration file. The default filename is `jrly.config`. If the configuration file does not exist or cannot be opened, the JRLY prints an error message. For information about JRLY error messages, see the *System Messages* in the WebLogic Enterprise online documentation.

If the JRLY is unable to start, it writes a message to `stderr` and attempts to log the startup failure in the error log (specified in the JRLY configuration file), and then exits.

JRLY Command-Line Options for Windows NT

This section describes command-line options that are available for the Windows NT version of `JRLY.exe`. Note the following:

- JRLY as a Windows NT service is available only for Windows NT.
- When the display suffix is optional (when [*display_suffix*] is shown), all operations are performed on the default JRLY Windows NT service instance.
- For manually installed, additional JRLY services, a suffix (any string) is required. Also, you can install the default service manually by omitting the optional string suffix.
- Each instance of JRLY Windows NT service uses the same binary executable file.
- A new process is started for each instance of JRLY Windows NT service.
- The syntax for these options is: `jrly -command.`
- Text specified within brackets ([ ]) is optional.
- All commands in Table 3-4, except for `-start` and `-stop`, require that you have write access to the Windows NT Registry.

- The `-start` and `-stop` commands require that you have Windows NT service control access. These requirements are based on Windows NT user restrictions.

Table 3-4 describes t**he JRLY command-line options**.

Table 3-4 JRLY Command-Line Options for Windows NT

Option	Description

`jrly -install [`*`display_suffix`*`]`	Installs `jrly` as a Windows NT service.
	Example 1:
	`jrly -install`
	In this example, the default JRLY is installed as a Windows NT service and is displayed in the Service Control Manager (SCM) as **Jolt Relay**.
	Example 2:
	`jrly -install MASTER`
	In this case, an instance of JRLY is installed as a Windows NT service and is displayed in the SCM as **Jolt Relay_MASTER**. The suffix, MASTER, does not have any significance; it is only used to uniquely identify various instances of JRLYs.
	At this point, this instance of JRLY is not ready to start. It must be assigned the configuration file (see the `set` command discussion) that specifies the listening TCP/IP port, JSH connection TCP/IP port, log files, and `SOCKETTIMEOUT`). This file should not be shared between various instances of JRLY.

`jrly -remove [`*`display_suffix`*`] | -all`	Removes one or all instances of JRLY from a Windows NT service. If `[`*`display_suffix`*`]` is specified, this command removes the specified JRLY service. If `[`*`display_suffix`*`]` is not specified, this command removes the default JRLY from being a Windows NT service. If the `-all` option is specified, all JRLY Windows NT services are removed. Related Windows NT registry entries are removed.

Table 3-4 JRLY Command-Line Options for Windows NT (Continued)

Option	Description

`jrly -set [-d display_suffix] -f config_file`	Updates the registry with the full path of a new configuration file.
	Example 1:
	`jrly -set -f c:\tux71\udataobj\jolt\jrly.config`
	In this example, the default JRLY Windows NT service (Jolt Relay) is assigned a configuration file called `jrly.config` that is located in `c:\tux71\udataobj\jolt` directory.
	Example 2:
	`jrly -set -d MASTER -f c:\tux71\udataobj\jolt\master.con`
	Here, the JRLY Windows NT service instance, called **Jolt Relay_MASTER** is assigned a configuration file called `jrly_master.con` that is located in `c:\tux71\udataobj\jolt` directory.

`jrly -manual [display_suffix]`	Sets the start/stop to manual. This command sets the specified JRLY instance to be manually controlled, using either the command-line options or the SCM.
`jrly -auto [display_suffix]`	Sets the start/stop to automatic. This command sets all the operations for the specified Windows NT service to be automatically started when the OS boots and stopped when the OS shuts down.
`jrly -start [display_suffix]`	Starts the specified JRLY.
`jrly -stop [display_suffix]`	Stops the specified JRLY.
`jrly -version`	Prints the current version of JRLY binary.
`jrly -help`	Prints command-line options with brief descriptions.

JRLY Command-Line Option for UNIX

Table 3-5 describes t**he one JRLY command-line option for UNIX**.

Table 3-5 JRLY Command-Line Option for UNIX

Option	Description
`jrly -f config_file`	Starts the JRLY process. This option starts the JRLY process with the specified configuration file (path and name). If the configuration file does not exist or cannot be opened, the JRLY prints an error message. If the JRLY cannot start, it writes a message to `stderr`, attempts to log the startup failure in the error log, then exits.

JRLY Configuration File

The format of the configuration file is a `TAG=VALUE` format. Blank lines or lines starting with the # character are ignored. Listing 3-3 contains an example of the formal specifications of the configuration file.

Listing 3-3 Specification of Configuration File

```
LOGDIR=<LOG_DIRECTORY_PATH>
ACCESS_LOG=<ACCESS_FILE_NAME in LOGDIR>
ERROR_LOG=<ERROR_FILE_NAME in LOGDIR>
LISTEN=<IP:Port combination where JRLY will accept connections>
CONNECT=<IP:Port combination associated with JRAD>

SOCKETTIMEOUT=<Seconds for socket accept()function>
```

`SOCKETTIMEOUT` is the duration (in seconds) of which the relay Windows NT service blocks the establishment of new socket connections to allow network activity (new connections, data to be read, closed connections). It is valid only on Windows NT machines. `SOCKETTIMEOUT` also affects the SCM. When the SCM requests that the service stop, the SCM needs to wait at least `SOCKETTIMEOUT` seconds before doing so.

Listing 3-4 shows an example of the JRLY configuration file. The `CONNECT` line specifies the IP address and port number of JRAD machine.

Listing 3-4 Example of JRLY Configuration File

```
LOGDIR=/usr/log/relay
ACCESS_LOG=access_log
ERROR_LOG=errorlog
# jrly will listen on port 4444
LISTEN=200.100.10.100:4444
CONNECT=200.100.20.200:4444, 200.100.20.200:5555,...

SOCKETTIMEOUT=30            //See text under listing
```

The format for directory and filenames is determined by the operating system. UNIX systems use the forward slash (/). Windows NT systems use the backslash (\). If any file specified in `LOGDIR`, `ACCESS_LOG` or `ERROR_LOG` cannot be opened for writing, the JRLY prints an error message on `stderr` and exits.

Table 3-6 describes t**he** formats for host names and port numbers.

Table 3-6 Host Name and Port Number Formats

Host Name/Port #	Description
Hostname:Port	*Hostname* is a string, *Port* is a decimal number.
//Hostname:Port	*Hostname* is a string, *Port* is a decimal number.
IP:Port	*IP* is a dotted notation IP address, *Port* is a decimal number.

Jolt Relay Adapter

The Jolt Relay Adapter (backend relay) is a BEA Tuxedo system server. The Jolt Relay Adapter (JRAD) server may or may not be located on the same BEA Tuxedo host machine in single host mode (SHM) and server group to which the JSL server is connected.

The JRAD can be started independently of its associated JRLY. JRAD tracks its startup and shutdown activity in the BEA Tuxedo log file.

JRAD Configuration

A single JRAD process can only be connected to a single JRLY. A JRAD can be configured to communicate with only one JSL and its associated JSHs. However, multiple JRADs can be configured to communicate with one JSL. The `CLOPT` parameter for the BEA Tuxedo servers must be included in the `UBBCONFIG` file. Listing 3-5 shows a sample of the file.

Table 3-7 describes additional information about the `CLOPT` parameters.

Table 3-7 JRAD CLOPT Parameter Descriptions

CLOPT Parameter	Description
`-l` *`hexadecimal format`*	Port to listen for the JRLY to connect on behalf of the client.
`-c` *`hexadecimal format`*	The address of the corresponding JSL to which JRAD connects.
`-H` *`hexadecimal format`*	Used when there is a network address translation performed for JRLY listen address.

Note: The format is `0x0002PPPPNNN`.

Listing 3-5 Sample JRAD Entry in UBBCONFIG File

```
# JRAD host 200.100.100.10 listens at port 2000, connects to JSL
port 8000 on the same host

JRAD    SRVGRP=JSLGRP   SRVID=60
        CLOPT="-A -- -l 0x000207D0C864640A –c 0x00021f40C864640A"
```

Network Address Configurations

A Jolt Internet Relay configuration requires that several networked components work together. Prior to configuration, review the criteria in Table 3-8 and record the information to minimize the possibility of misconfiguration.

Table 3-8 Jolt Internet Relay Network Address Configuration Criteria

JRLY	JRAD	JSL
LISTEN: *Location where the clients connect*	-l: *Location where the listener connects to the JRLY*	-n: *Location of JSL.* Must match -c parameter of JRAD
CONNECT: *Location of your JRAD*. Must match the -l parameter of JRAD	-c: *Location of JSL*. Must match -n parameter of JSL	

4 Using the Bulk Loader Program

This topic includes the following sections:

- Defining Bulk Loader Data Files
- Running the Bulk Loader
- Troubleshooting

As a systems administrator, you may have an existing BEA Tuxedo application with multiple BEA Tuxedo services. Manually creating these definitions in the Jolt Repository database may take a long time to complete. The Jolt Bulk Loader is a command utility that allows you to load multiple, previously defined BEA Tuxedo services to the Jolt Repository database in a single step. Using the `jbld` program, the Bulk Loader utility reads the BEA Tuxedo service definitions from a text file (that you create according to a specific format) and loads them into the Jolt Repository. The service definitions are loaded to the Repository database in one *bulk load*. After the services populate the Jolt Repository, you can create, edit, and group services using the Jolt Repository Editor.

Note: In order to use the Bulk Loader to add service definitions, you must first configure the Jolt Servers (JSL and JSHs). If you want to use this tool across a firewall, you must also configure Jolt Internet Relay. For configuration instructions, see Chapter 3, “Configuring JET for Client Access.”

Defining Bulk Loader Data Files

This topic includes the following sections:

- About Bulk Loader Data Files
- Guidelines for Using Keywords
- Keyword Order in the Bulk Loader Data File
- Using Service-level Keywords and Values
- Sample Bulk Loader Data File

About Bulk Loader Data Files

The bulk loader data file is a text file that defines BEA Tuxedo services and their associated parameters. You create this text file (using a text editor) in accordance with the syntax rules described later in this topic.

The Bulk Loader loads the services defined in the bulk loader data file into the Jolt Repository using the package name `BULKPKG` by default. The `-p` command overrides the default and you can give the package any name you choose. If another load is performed from a bulk loader data file with the same `-p` option, all the services in the original package are deleted and a new package is created with the services from the new bulk loader data file.

If a service exists in a package other than the package you name that uses the `-p` option, the Bulk Loader reports the conflict and does not load a service from the bulk loader Data file into the Repository. Use the Jolt Repository Editor to remove duplicate services and load the bulk loader Data file again. See “Using the BEA Jolt Repository Editor” on page 5-1 for additional information.

Each service definition consists of service properties and parameters that have a set number of parameter properties. Each property is represented by a keyword and a value. Keywords are divided into two levels:

- Service-level
- Parameter-level

Guidelines for Using Keywords

The `jbld` program reads the service definitions from a text file. To use the keywords, observe the guidelines in Table 4-1.

Table 4-1 Guidelines for Using Keywords

Guideline	Example
Each keyword must be followed by an equal sign (=) and the value	**Correct:** `type=string` **Incorrect:** `type`
Only one keyword is allowed on each line	**Correct:** `type=string` **Incorrect:** `type=string access=out`
Any lines not having an equal sign (=) are ignored	**Correct:** `type=string` **Incorrect:** `type string`
Certain keywords only accept a well-defined set of values	The keyword `access` accepts only these values: `in`, `out`, `inout`, `noaccess`

The input file can contain multiple service definitions	`service=INQUIRY`
	`<service keywords and values>`
	`service=DEPOSIT`
	`<service keywords and values>`
	`service=WITHDRAWAL`
	`<service keywords and values>`
	`service=TRANSFER`
	`<service keywords and values>`

Each service definition consists of multiple keywords and values	`service=DEPOSIT`
	`export=true`
	`inbuf=VIEW32`
	`outbuf=VIEW32`
	`inview=INVIEW`
	`outview=OUTVIEW`

Keyword Order in the Bulk Loader Data File

Keyword order must be maintained within the data files to ensure an error-free transfer during the bulk load.

The first keyword definition in the bulk loader data text file must be the initial `service=<NAME>` keyword definition (shown in Listing 4-1). Following the `service=<NAME>` keyword, all remaining service keywords that apply to the named service must be specified before the first `param=<NAME>` definition. These remaining service keywords can be in any order.

All parameters associated with the service must be specified. Following each `param=<NAME>` keywords are all the parameter keywords that apply to the named parameter until the next occurrence of a parameter definition. These remaining parameter keywords can be in any order. When all the parameters associated with the first service are defined, specify a new `service=<NAME>` keyword definition.

Listing 4-1 Keyword Hierarchical Order in a Data File

```
service=<NAME>
<service keyword>=<value>
<service keyword>=<value>
<service keyword>=<value>
param=<NAME>
<parameter keyword>=<value>
<parameter keyword>=<value>
param=<NAME>
<parameter keyword>=<value>
<parameter keyword>=<value>
```

Using Service-level Keywords and Values

A service definition must begin with the `service=<NAME>` keyword. Services using CARRAY or STRING buffer types should only have one parameter in the service. The recommended parameter name for a service that uses a `CARRAY` buffer type is `CARRAY` with `carray` as the data type. For a service that uses a `STRING` buffer type, the recommended parameter name is `STRING` with `string` as the data type.

Table 4-2 describes the guidelines for use of the service-level keywords and acceptable values for each.

Table 4-2 Service-level Keywords and Values

Keyword	Value
`service`	Any BEA Tuxedo service name.
`export`	Either `true` or `false` (default is `false`).

`inbuf/outbuf`	Select one of these buffer types:
	`FML`
	`FML32`
	`VIEW`
	`VIEW32`
	`STRING`
	`CARRAY`
	`X_OCTET`
	`X_COMMON`
	`X_C_TYPE`
	For more information, see “Using BEA Tuxedo Buffer Types with JET” on page 1-13.

`inview`	Any view name for input parameters. (This keyword is optional *only* if one of the following buffer types is used: `VIEW`, `VIEW32`, `X_COMMON`, `X_C_TYPE`.)
`outview`	Any view name for output parameters. (Optional)

Using Parameter-level Keywords and Values

A parameter begins with the `param=<NAME>` keyword followed by a number of parameter keywords. It ends when another `param` or `service` keyword, or end-of-file is encountered. The parameters can be in any order after the `param=<NAME>` keyword.

Table 4-3 contains the guidelines for use of the parameter-level keywords and acceptable values for each.

Table 4-3 Parameter-level Keywords and Values

Keyword	Values
`param`	Any parameter name.

`type`	`byte`
	`short`
	`integer`
	`float`
	`double`
	`string`
	`carray`

`access`	`in`
	`out`
	`inout`
	`noaccess`

`count`	Maximum number of occurrences (default is 1). The value for unlimited occurrences is 0. Used only by the Jolt Repository Editor to format test screens.

Sample Bulk Loader Data File

Listing 4-2 contains a sample data file in the correct format using the UNIX command `cat servicefile`. This sample loads `TRANSFER`, `LOGIN`, and `PAYROLL` service definitions to the `BULKPKG`.

Listing 4-2 Sample Bulk Loader Data File

```
service=TRANSFER
export=true
inbuf=FML
outbuf=FML
param=ACCOUNT_ID
type=integer
access=in
count=2
param=SAMOUNT
type=string
access=in
param=SBALANCE
type=string
access=out
count=2
param=STATLIN
type=string
access=out

service=LOGIN
inbuf=VIEW
inview=LOGINS
outview=LOGINR
export=true
param=user
type=string
access=in
param=passwd
type=string
access=in
param=token
type=integer
access=out

service=PAYROLL
inbuf=FML
outbuf=FML
```

```
param=EMPLOYEE_NUM
type=integer
access=in
param=SALARY
type=float
access=inout
param=HIRE_DATE
type=string
access=inout
```

Running the Bulk Loader

The `jbld` program is a Java application. Before running the `jbld` command, set the `CLASSPATH` environment variable (or its equivalent) to point to the directory where the Jolt class directory (that is, `jolt.jar` and `joltadmin.jar`) is located. If the `CLASSPATH` variable is not set, the Java Virtual Machine (JVM) cannot locate any JET classes.

Note: If the `m3.jar` file is already specified in the `CLASSPATH`, then the directory containing the `jolt.jar` and `joltadmin.jar` files must be specified *ahead of* the directory containing the `m3.jar` file.

The Bulk Loader utility gets its input from command-line arguments and from the input file. For security reasons, `jbld` does not use command-line arguments to specify user authentication information (user password or application password). Depending on the server's security level, `jbld` automatically prompts the user for passwords.

To run the Bulk Loader:

1. Type the following at the prompt (with the correct options):

   ```
   java bea.jolt.admin.jbld [-n][-p package][-u usrname][-r
   usrrole] //host:port filename
   ```

2. Use the command-line options described in Table 4-4.

Table 4-4 Bulk Loader Command-line Options

Option	Description
`-u` *`usrname`*	Specifies the username (default is your account name). (Mandatory, if required by security)
`-r` *`usrrole`*	Specifies the user role (default is `admin`). (Mandatory, if required by security)
`-n`	Validates input file against the current Repository; no updates are made to the Repository. (Optional)
-p package	Repository package name (default is `BULKPKG`).
`//host:port`	Specifies the JRLY or JSL address (host name and IP port number). (Mandatory)
`filename`	Specifies the file containing the service definitions. (Mandatory)

Troubleshooting

See Table 4-5 if you encounter problems using the Bulk Loader utility.

Table 4-5 Bulk Loader Troubleshooting Table

If . . .	Then . . .
The data file is not found	Check to ensure that the path is correct
The keyword is invalid	Check to ensure that the keyword is valid for the package, service, or parameter
The value of the keyword is null	Type a value for the keyword
The value is invalid	Check to ensure that the value of a parameter is within the allocated range for that parameter

Table 4-5 Bulk Loader Troubleshooting Table (Continued)

If . . .	Then . . .
The data type is invalid	Check to ensure that the parameter is using a valid data type

5 Using the BEA Jolt Repository Editor

Use the BEA Jolt Repository Editor to add, modify, test, export, and delete BEA Tuxedo service definitions from the Jolt Repository based on the information available from the BEA Tuxedo configuration file. The Jolt Repository Editor accepts BEA Tuxedo service definitions, including the names of the packages, services, and parameters.

Note: Before you use the Jolt Repository Editor, you must first configure the JET system according to the instructions in Chapter 3, "Configuring JET for Client Access." In particular, the JSL and JSHs must be correctly configured and running.

This topic includes the following sections:

- Introducing the Jolt Repository Editor
- Getting Started with the Jolt Repository Editor
- Main Components of the BEA Jolt Repository Editor
- Setting Up Packages and Services
- Grouping Services Using the Package Organizer
- Modifying Packages, Services, and Parameters
- Making a Service Available to the JET Client
- Testing a Service
- Troubleshooting

- Repository Enhancements for Jolt

Note: In order to use the Jolt Repository Editor to edit service definitions, you must first configure the Jolt Servers (JSL and JSHs). If you want to use this tool across a firewall, you must also configure Jolt Internet Relay. For configuration instructions, see Chapter 3, "Configuring JET for Client Access."

Introducing the Jolt Repository Editor

This topic includes the following sections:

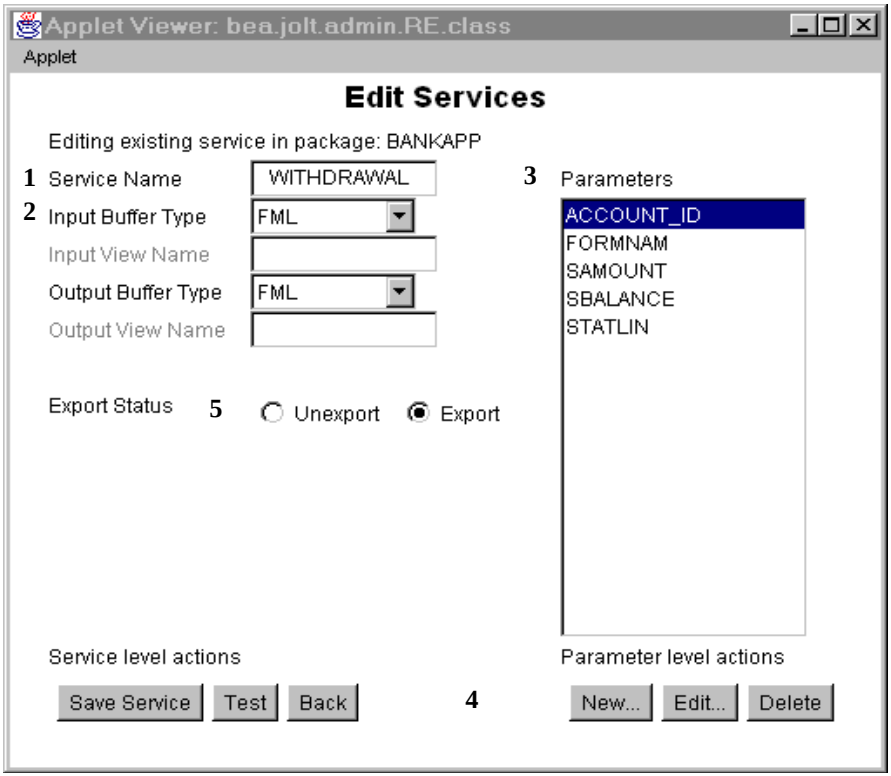
- Jolt Repository Editor Window
- Components of the Jolt Repository Editor Window

The Jolt Repository is used internally by JET to translate data between JET and BEA Tuxedo type buffers. The Jolt Repository Editor is used to edit service definitions in the Repository. The Jolt Repository Editor is available as a downloadable Java applet. When a BEA Tuxedo service is added to the Repository, it must be exported to the Jolt server to ensure that the client requests can be made from a JET client.

Jolt Repository Editor Window

Jolt Repository Editor windows contain entry fields, scrollable displays, command buttons, status, and radio buttons. Figure 5-1 illustrates the parts of the window. Table 5-1 describes each component.

Figure 5-1 Jolt Repository Editor Window



Components of the Jolt Repository Editor Window

Table 5-1 describes the parts of the Jolt Repository Editor window shown in Figure 5-1.

Table 5-1 Jolt Repository Editor Window Components

#	Component	Description
1	Text boxes	Enter text, numbers, or alphanumeric characters such as `Service Name`, `Input View Name`, server names, or port numbers. In Figure 5-1, `Service Name` is used.
2	Drop-down arrow	View lists that extend beyond the display using an arrow button. In Figure 5-1, `Input Buffer Type` is used.
3	Display list	Select from a list of predefined items such as the Parameters list or select from a list of items that have been defined.
4	Command buttons	Activate an operation such as displaying the Packages window, Services window, or Package Organizer. In Figure 5-1, command buttons include: Save Service, Test, Back, New, Edit, and Delete.
5	Radio buttons	Select one of a number of options. Only one of the radio buttons can be activated at a time. For example, Export Status can only be `Unexport` or `Export`.

Getting Started with the Jolt Repository Editor

Note: Before you use the Jolt Repository Editor, you must first configure the JET system according to the instructions in Chapter 3, "Configuring JET for Client Access." In particular, the JSL and JSHs must be correctly configured and running.

To use the Jolt Repository Editor:

1. Start the Jolt Repository Editor.

   You can start the Jolt Repository Editor from either the JavaSoft `appletviewer` or from your Web browser. Both of these methods are detailed in the following sections.

2. Log on to the Jolt Repository Editor.

Note: For information about exiting the Jolt Repository Editor after you enter information, see "Exiting the BEA Jolt Repository Editor" on page 5-9.

Starting the Jolt Repository Editor

You can start the Jolt Repository Editor from either the JavaSoft `appletviewer` or from your Web browser.

Starting the Jolt Repository Editor Using the Java Applet Viewer

To start the Jolt Repository Editor using the Java applet viewer:

1. Set the `CLASSPATH` to include the Jolt class directory or the directory in which the `*.jar` files reside (such as the `jolt.jar` file under `udataobj\jolt`).

   Note: If the `m3.jar` file is already specified in the `CLASSPATH`, then the directory containing the `jolt.jar` file must be specified *ahead of* the directory containing the `m3.jar` file.

2. If you are loading the applet from a local disk, set the default directory to the directory where the `RE.html` file resides (such as `udataobj\jolt`), and then type the following at the URL location:

   ```
   appletviewer RE.html
   ```

   If you are loading the applet from the Web server, type the following at the URL location (including the full path):

   ```
   appletviewer http://<www.server>/<URL path>/RE.html
   ```

3. Press Enter.

   The window is displayed as shown in Figure 5-2.

Starting the Jolt Repository Editor from Your Web Browser

From a Web browser, you can start the Jolt Repository Editor from a local file or from a Web server.

Starting from a Local File

To start the Jolt Repository Editor from a local file:

1. Set the `CLASSPATH` to include the Jolt class directory or the directory in which the `*.jar` files reside (such as the `jolt.jar` file under `udataobj\jolt`).
2. Set the default directory to the directory where the `RE.html` file resides (such as `udataobj\jolt`).
3. Type the following command:

   ```
   file:RE.html
   ```

4. Press Enter.

   The BEA Jolt Repository Editor Login window appears, as shown in the example in Figure 5-2.

Starting from a Web Server

To start the Jolt Repository Editor from a Web server:

1. Ensure that the `CLASSPATH` does *not* include the Jolt class directory.
2. Remove the Jolt classes from the `CLASSPATH`.

3. Type the following, including the full path:

   ```
   http://<www.server>/<URL path>/RE.html
   ```

   Note: If `jolt.jar` and `admin.jar` are in the same directory as `RE.html`, the Web server provides the classes. If they are not in the same directory as `RE.html`, modify the `applet codebase` parameter in the `Re.html` file.

4. Press Enter.

   The BEA Jolt Repository Editor Login window appears, as shown in the example in Figure 5-2.

Logging On to the Jolt Repository Editor

After starting the Jolt Repository Editor, complete the following steps to log on:

Note: The Jolt Repository Editor Logon Window must be displayed before you log on. See Figure 5-2 as you perform the following procedure.

1. In the logon window, type the name of the Server machine designated as the "access point" to the BEA Tuxedo application, and then press Tab.
2. Type the Port Number and press Enter.

   The system validates the server and port information.

   Note: Unless you are logging on through Jolt Relay, the same port number is used to configure the Jolt Listener. See your `UBBCONFIG` file for additional information.

3. Type the BEA Tuxedo Application Password and press Enter.

   Depending upon the authentication level, complete steps 4 and 5 as required.

4. Type the BEA Tuxedo User Name and press Tab.
5. Type the BEA Tuxedo User Password and press Enter.

   The Packages and Services command buttons are enabled.

Sample Logon Window

Figure 5-2 BEA Jolt Repository Editor Logon Window

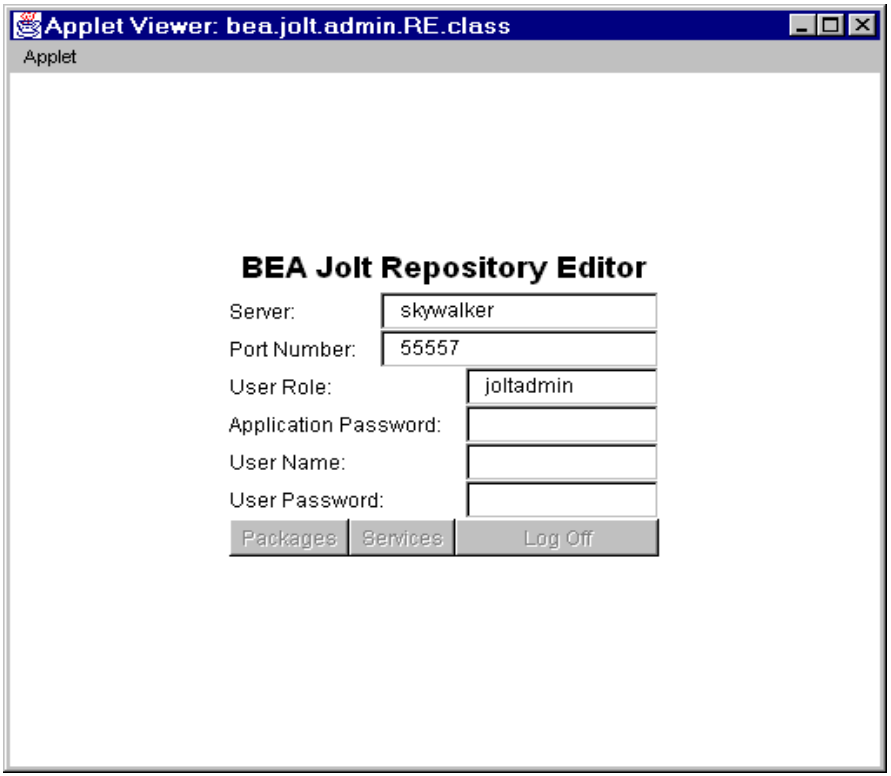


Table 5-2, describes the Jolt Repository Editor logon window elements.

Components of the BEA Jolt Repository Editor Logon Window

Table 5-2 describes the components of the BEA Jolt Repository Editor Logon window shown in Figure 5-2.

Table 5-2 BEA Jolt Repository Editor Logon Window Components

Option	Description
Server	Server name.

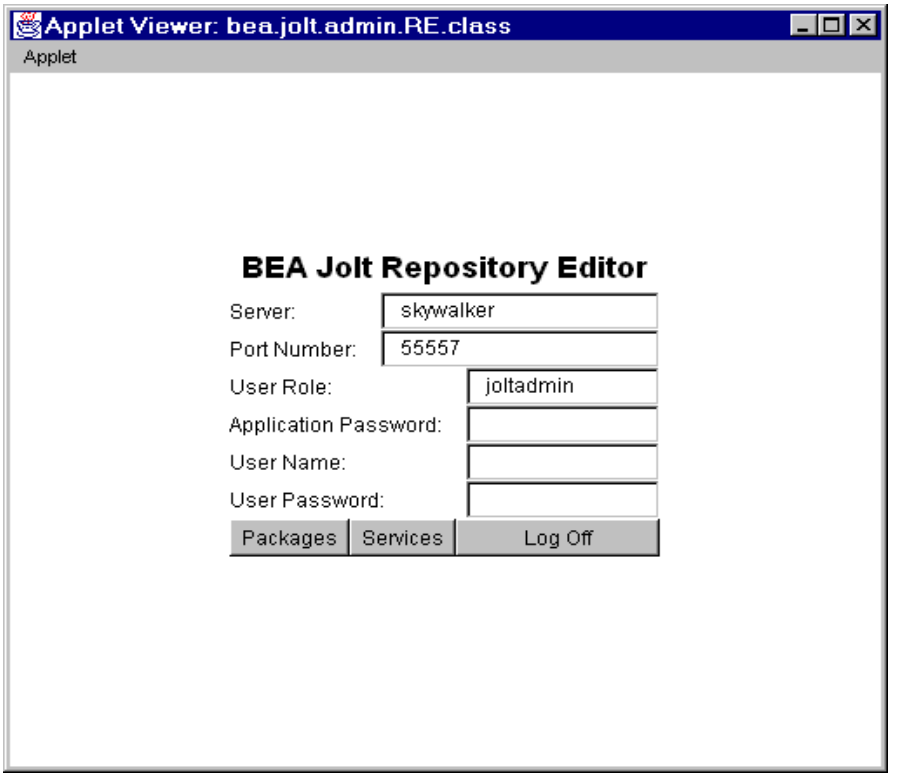
Table 5-2 BEA Jolt Repository Editor Logon Window Components (Continued)

Option	Description
Port Number	Port number in decimal value. **Note:** After the Server Name and Port Number are entered, the User Name and Password fields are activated. Activation is based on the authentication level of the BEA Tuxedo application.
User Role	BEA Tuxedo user role. Required only if BEA Tuxedo authentication level is USER_AUTH or higher.
Application Password	BEA Tuxedo administrative password text entry.
User Name	BEA Tuxedo user identification text entry. The first character must be an alpha character.
User Password	BEA Tuxedo password text entry.
Packages	Accesses the Packages window. (Enabled after the logon.)
Services	Accesses the Services window. (Enabled after the logon.)
Log Off	Terminates the connection with the server.

Exiting the BEA Jolt Repository Editor

Exit the BEA Jolt Repository Editor when you finish adding, editing, testing, or deleting packages, services, and parameters. Prior to exiting, the window is displayed as shown in Figure 5-3.

Figure 5-3 BEA Jolt Repository Editor Logon Window Prior to Exiting



Note that only the Packages, Services, and Log Off command buttons are enabled. All of the text entry fields are disabled.

To exit the Jolt Repository Editor:

1. Click Back to return to the Jolt Repository Editor Logon window.
2. Click Log Off to terminate the connection with the server.

   The Jolt Repository Editor Logon window shows disabled fields.
3. Click Close from your browser menu to close the window.

Main Components of the BEA Jolt Repository Editor

This topic includes the following sections:

- Workflow for the BEA Jolt Repository Editor
- What Is a Package?
- What Is a Service?
- Working with Parameters

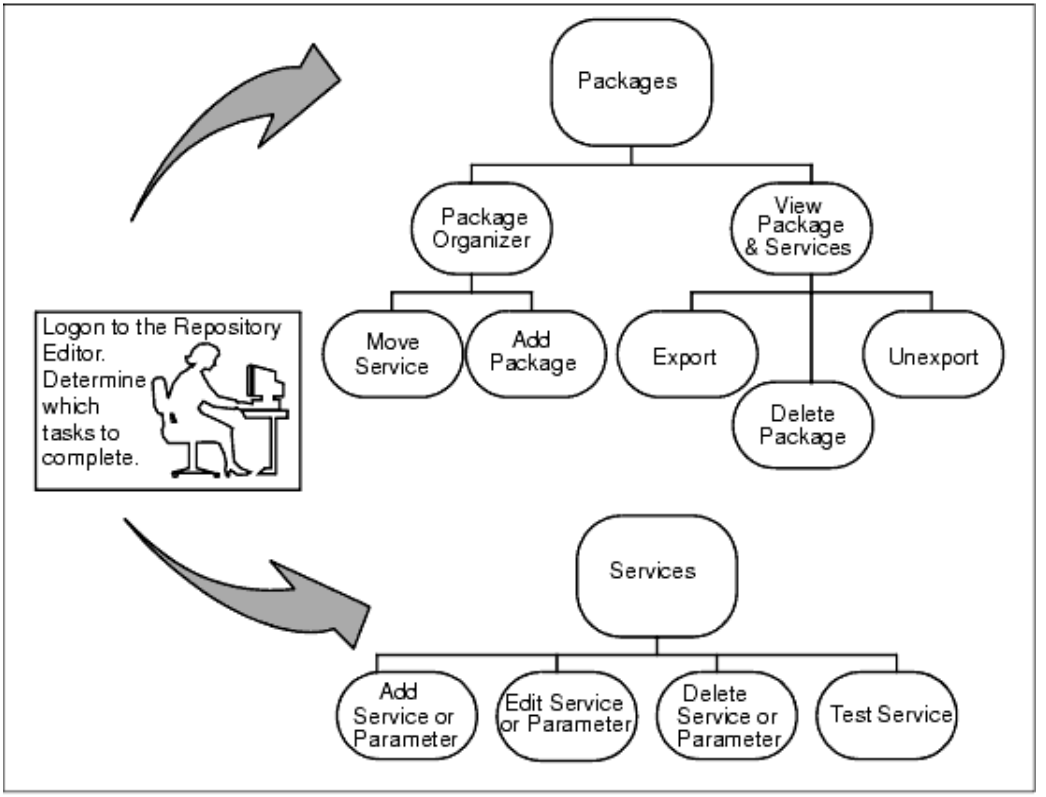
The Jolt Repository Editor allows you to add, modify, or delete any of the following components: packages, services (you can also test and group services), and parameters.

Workflow for the BEA Jolt Repository Editor

After you log on to the Jolt Repository Editor, two buttons are enabled, Packages and Services.

Figure 5-4 illustrates the Jolt Repository Editor flow to help you determine which of these two buttons to select.

Figure 5-4 Jolt Repository Editor Flow Diagram



Select Packages to open the Packages window and perform the following functions:

- View packages and services
  - Make a service available using Export or Unexport
  - Select a package to delete
- Access the Package Organizer to:
  - Move services from one package to another
  - Create a new package

See “What Is a Package?” on page 5-13 for complete details.

Use Services to open the Services window and perform the following functions:

- Create, add, edit, or delete service definitions
- Create, add, edit, or delete parameters
- Test the services and parameters

See “What Is a Service?” on page 5-16 for complete details.

What Is a Package?

Packages provide a convenient method for grouping services for Jolt administration.

You use the Packages window to perform the following tasks:

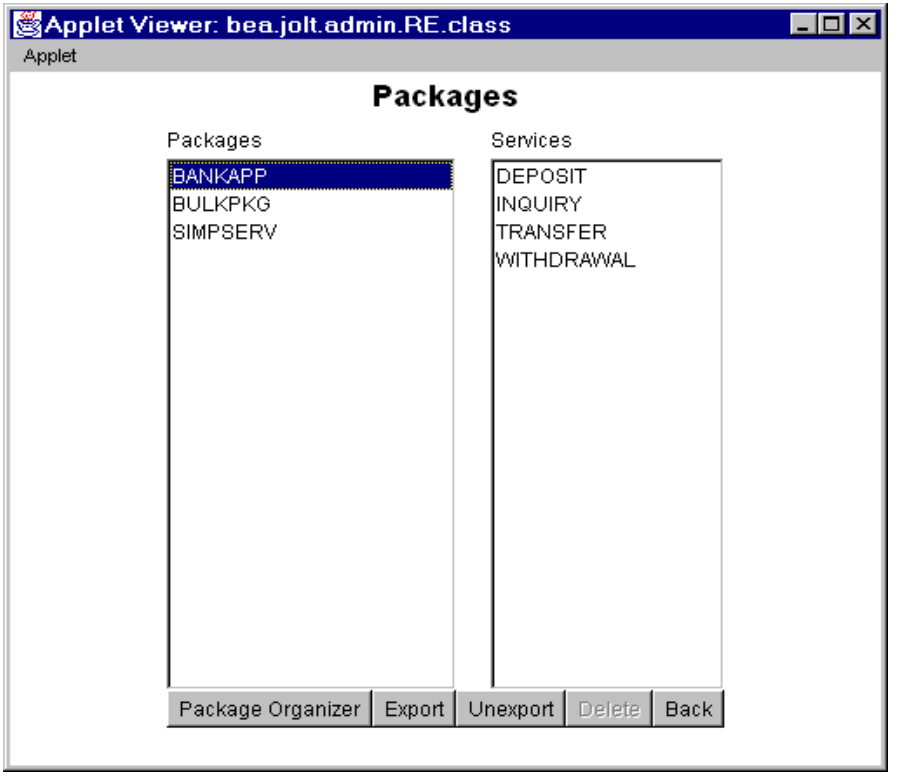
- View packages and services.
- Export or unexport services.
- Delete packages.
- Access Package Organizer to:
  - Move services
  - Create a new package

Click the Packages button in the Jolt Repository Editor logon window to display the available packages. When you select a specific package from the display list, its services within that package are displayed.

Sample Packages Window

Figure 5-5 contains a sample Packages window.

Figure 5-5 Packages Window



Components of the Packages Window

Table 5-3 describes the components of the Packages window shown in Figure 5-3.

Table 5-3 Packages Window Components

Option	Description
Packages	Lists available packages.
Services	Lists available services within the selected package.
Package Organizer	Accesses the Package Organizer window to review available packages and services. Use this window to move the services among the packages or add a new package.

Table 5-3 Packages Window Components (Continued)

Option	Description
Export	Makes the most current services available to the client. This option is enabled when a package is selected.
Unexport	Select this option before testing an existing service. This option is enabled when a package is selected.
Delete	Deletes a package. This option is enabled when a package is selected and the package is empty (no services contained within the package).
Back	Returns the user to the previous window.

Viewing a Package

To view a package:

1. Click Packages in the Jolt Repository Editor Logon window.

   The Packages window opens and displays the list of available packages.

   In Figure 5-5, `BANKAPP`, `BULKPKG`, and `SIMPSERV` are the available packages.

2. See "Viewing a Parameter" on page 5-19 for additional information.

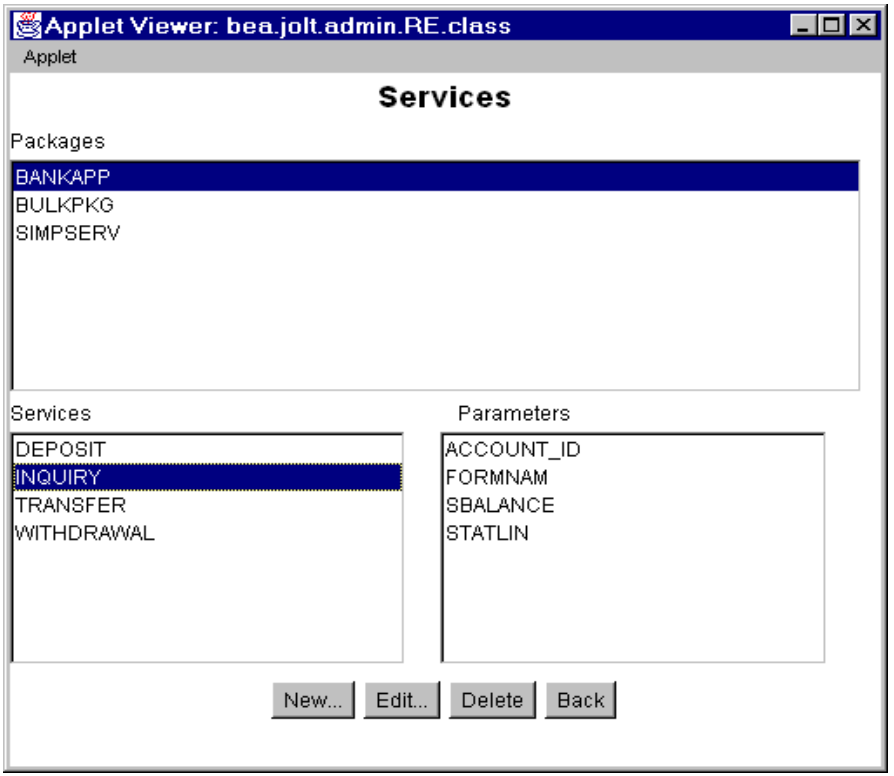
What Is a Service?

A *service definition* describes the properties of a BEA Tuxedo service, such as its name, input and output buffer types, and individual parameters. Adding or editing a Jolt service in the Jolt Repository does *not* change an existing BEA Tuxedo service.

Sample Services Window

You use the Services Window to add, edit, or delete services. Figure 5-6 shows an example of a Services window with the BANKAPP package selected, and the display list of services and parameters available for this package (parameters are detailed later).

Figure 5-6 Services Window



Components of the Services Window

Table 5-4 describes the components in the Services window shown in Figure 5-4.

Table 5-4 Services Window Description

Option	Description
Packages	Lists the available packages.
Services	Lists the services in the selected package, which you can edit or delete. Selecting a service displays the parameters within the service.
Parameters	Displays the parameters of the selected service.
New	Displays the Edit Services window for adding a new service.
Edit	Displays the Edit Services window for editing an existing service. This button is enabled only if a service has been selected.
Delete	Deletes a service. This button is only enabled if a service has been selected.
Back	Returns the user to the previous window.

Viewing a Service

To view a service:

1. Select Services from the Jolt Repository Editor Logon window.

   The Services window opens and displays the list of available packages.

2. Select a package.

   The list of available services for the selected package is displayed.

   In Figure 5-6, `BANKAPP` is the selected package. `DEPOSIT`, `INQUIRY`, `TRANSFER`, and `WITHDRAWAL` are the available services for `BANKAPP`.

3. See “Viewing a Parameter” on page 5-19 for additional information.

Working with Parameters

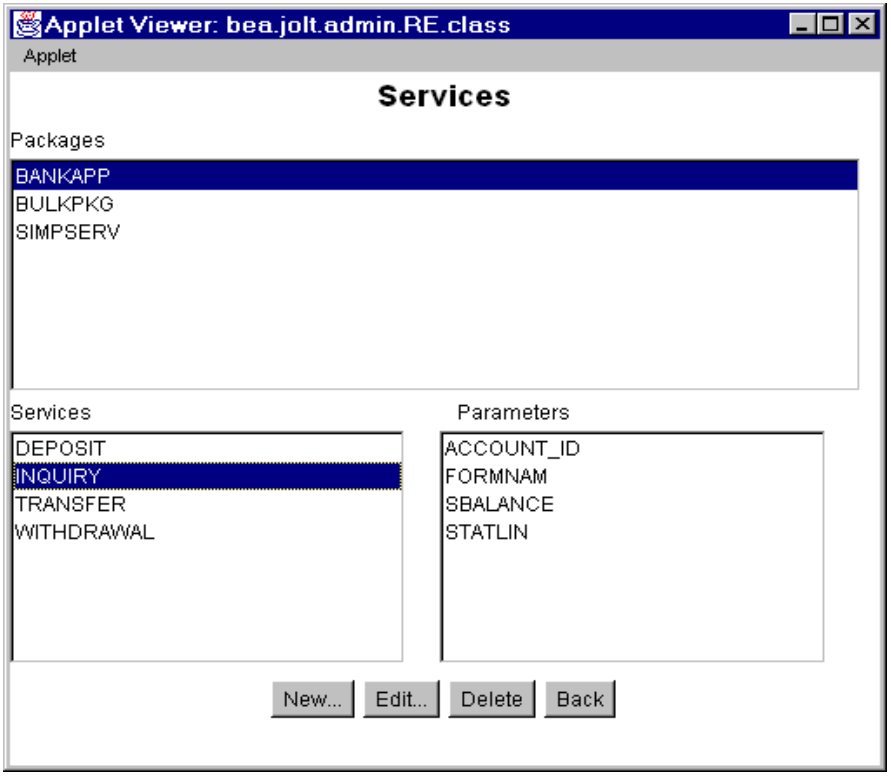
A service contains parameters, which may be a pin number, account number, payment, rate, term, age, or Social Security number.

Sample Services Window with Parameters

Figure 5-7 shows a Services window displaying a selected service and its parameters.

Note: Adding or editing a parameter does not modify or change an existing BEA Tuxedo Service.

Figure 5-7 Services Window with Parameters List



Viewing a Parameter

To view a parameter:

1. Select Services from the Jolt Repository Editor Logon window.

   The Services window opens and displays the list of available packages.

2. Select a package.

   The list of available services for the selected package is displayed.

   In Figure 5-7, `BANKAPP` is the selected package.

3. Select a service.

   The list of available parameters for the selected service is displayed.

   In Figure 5-7, `INQUIRY` is the selected service.

4. View the parameters for a selected service in the Parameters display list.

   In Figure 5-7, `ACCOUNT_ID`, `FORMNAM`, `SBALANCE,` and `STATLIN` are the available parameters for the `INQUIRY` service.

5. See "Adding Parameters" on page 5-26 for additional information.

Setting Up Packages and Services

This topic includes the necessary steps for setting up a package and its services:

- Saving Your Work
- Adding Packages
- Adding Services
- Adding Parameters

Saving Your Work

As you create and edit services and parameters, it is important to regularly save information to avoid losing input. Clicking Save Service in the Edit Services window can prevent the need to re-enter information in the event of a system failure.

Caution: When you add or edit the parameters of a service, you must select Add before choosing Back from the Edit Parameters window and returning to the Edit Services window.

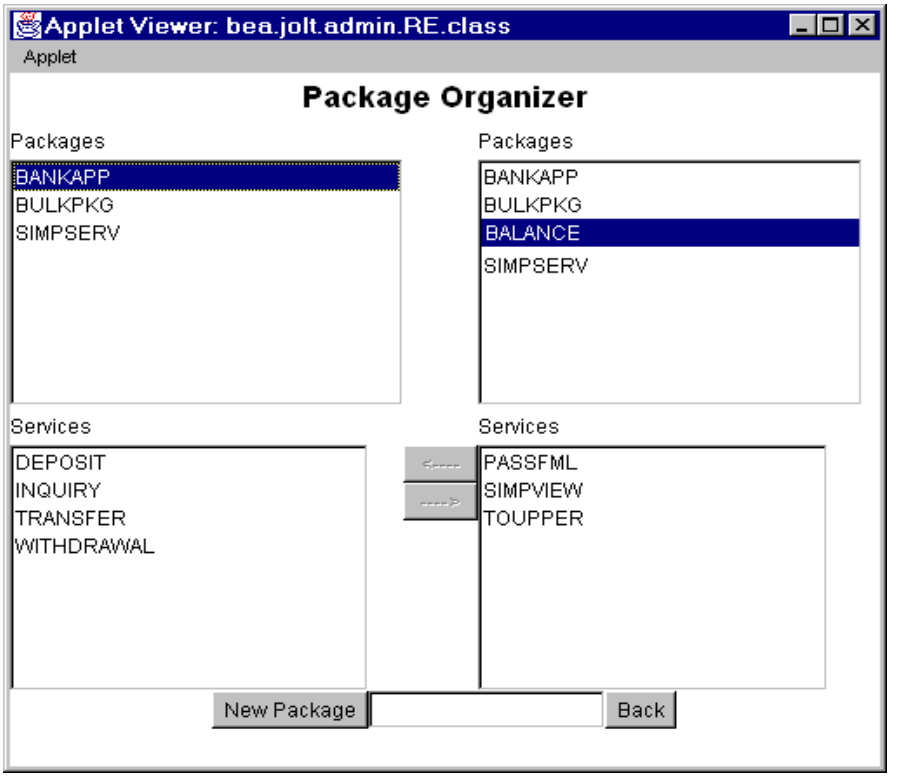
If adding a new service or modifying an existing service in the Edit Services window, be sure to select Save Service before choosing Back. If you select Back before you save the modified information, a warning is briefly displayed on the status line at the bottom of the window.

Adding Packages

When you need to add a new group of services, you create a new package before adding the services. The "Package Organizer Window" on page 5-21 and the following procedure show how to add a new package, `BALANCE`, to the Packages listing.

Sample Package Organizer Window

Figure 5-8 Package Organizer Window



Adding a Package

To add a package:

1. Click Packages in the Jolt Repository Editor Logon window to display the Packages window.
2. Select Package Organizer to display the Package Organizer window, similar to the one shown in Figure 5-8.

   For a description of contents of this window, see Table 5-7.
3. Click the New Package button in the Package Organizer window.

The text field is activated.

4. Type the name of the new package (not to exceed 32 characters) and press Enter.

   The new name (shown in Figure 5-8 as `BALANCE`) is displayed on the Packages list in random order.

Adding Services

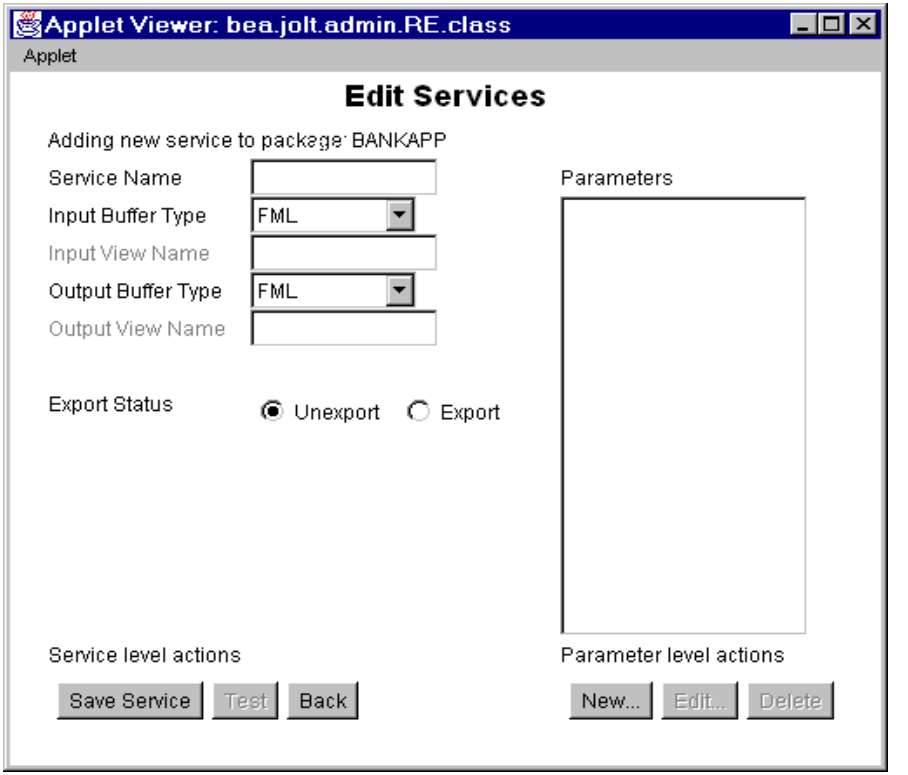
Services are definitions of available BEA Tuxedo services and can only be a part of a Jolt package.You must create the service as a part of a new or existing package.

The Jolt Repository Editor accepts the new service name exactly as it is typed (that is, all uppercase letters, abbreviations, misspellings are accepted). Service names must not exceed 30 characters.

Sample Edit Services Window

Figure 5-9 shows the Edit Services window for adding a service.

Figure 5-9 Edit Services Window: Add a New Service to a Package



Options for Adding a Service

Table 5-5 describes the options for adding services to a package in a package window.

Table 5-5 Options for Adding a Service

Option	Description

Table 5-5 Options for Adding a Service (Continued)

Edit Services Selections	Service Name	Name of the new service to be added to the Jolt Repository.
	Input Buffer Type/Output Buffer Type	■ VIEW— C-structure and 16-bit integer field. Contains subtypes that have a particular structure. X_C_TYPE and X_COMMON are equivalent. X_COMMON is used for COBOL and C. ■ VIEW32—similar to VIEW, except 32-bit field identifiers are associated with VIEW32 structure elements. ■ CARRAY—array of uninterrupted binary data that is neither encoded nor decoded during transmission; it may contain null characters. X_OCTET is equivalent. ■ FML—type in which each field carries its own definition. ■ FML32—similar to FML except the ID field and length field are 32 bits long. ■ STRING—character array terminated by a null character that is encoded or decoded. For more information, see "Using BEA Tuxedo Buffer Types with JET" on page 1-13.
	Input View Name/Output View Name	Unique name assigned to the Input View Buffer and Output View Buffer types. These fields are only enabled if VIEW or VIEW32 are the selected buffer types.
Export Status	Unexport Export	Radio button with current status of the service. EXPORT or UNEXPORT status is checked. UNEXPORT is the default.
Service Level Actions	Save Service	Saves the newly created service in the Repository.
	Test	Tests the service. This command button is disabled until a new service is created or edits to an existing service are saved.
	Back	Returns you to the previous window.
Parameter	Parameters	List of service parameters to edit or delete.
Parameter Level Actions	New	Adds new parameters to the service.
	Edit	Edits an existing parameter. This command button is disabled until a new parameter is selected.
	Delete	Deletes a parameter. This option is disabled until a parameter is selected.

Adding a Service

To add a service:

1. Select Services from the Jolt Repository Editor Logon window.

   The Services window opens, similar to the one shown in Figure 5-6.

2. Select the package to which you will add the service.

   If you later decide that another package should contain the new service, use the Package Organizer to move the service to a different package. (See "Grouping Services Using the Package Organizer" on page 5-31 for additional information.)

3. From the Services window, select New to display the Edit Services window, as shown in Figure 5-9.

4. Select the Service Name text field to activate it.

5. Type the name of the new service you want to add.

6. Select the input buffer type.

   Although the same buffer type selected for the Input Buffer is automatically selected for the Output Buffer, you can select a different Output Buffer type.

   - If `VIEW` or `VIEW32` is selected, you must type the Input View Name and Output View Name in the associated text fields.
   - If another buffer type is selected, the Input View Name and Output View Name text fields are disabled.
   - If `CARRAY` or `STRING` is selected, see "Selecting CARRAY or STRING as a Service Buffer Type" on page 5-25 for additional instructions.

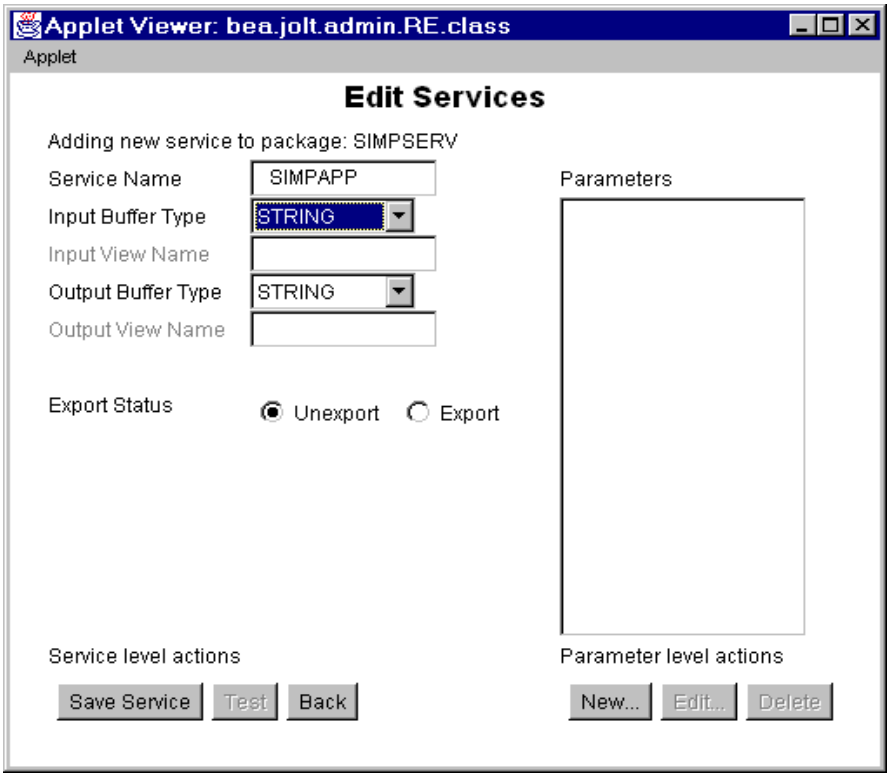
7. Select Save Service to save the newly created service.

Selecting CARRAY or STRING as a Service Buffer Type

If `CARRAY` or `STRING` is selected as the buffer type for a new service, only `CARRAY` or `STRING` can be added as the data type for the accompanying parameters. See also "Adding Parameters" on page 5-26 and "Selecting CARRAY or STRING as a Parameter Data Type" on page 5-29. For additional information, see Chapter 2, "Invoking BEA Tuxedo Services."

Figure 5-10 shows an example Edit Services window with `STRING` selected as the buffer type for the service `SIMPAPP`.

Figure 5-10 Edit Services Window: Select STRING Buffer Type



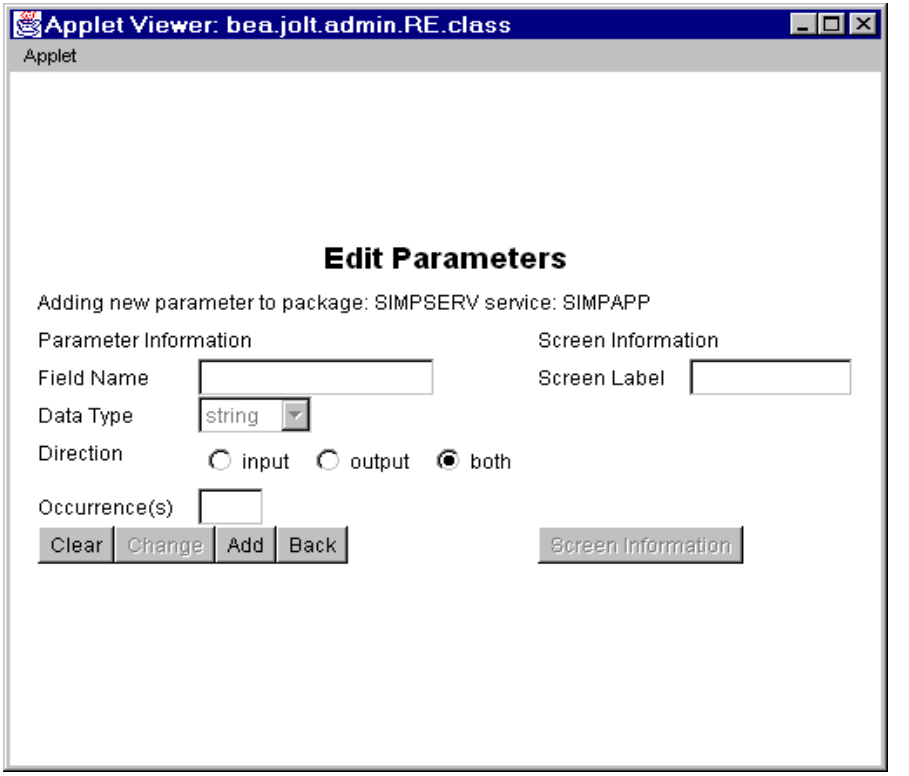
Adding Parameters

Clicking New under the label Parameter level actions in the Edit Services window displays the Edit Parameters window.

Sample Edit Parameters Window

Figure 5-11 shows an example of the Edit Parameters window, which you use to enter the parameter and screen information for a service.

Figure 5-11 Edit Parameters Window: Add a Parameter



Components of the Adding a Parameter Window

Table 5-6 describes the components of the Parameter window shown in Figure 5-11.

Table 5-6 Parameter Window Components

Option	Description
Field Name	Adds the field name (for example, asset).
Data Type	Lists data type choices: ■ byte—8-bit ■ short—16-bit ■ integer—32-bit ■ float—32-bit ■ double—64-bit ■ string—null-terminated character array ■ carray—variable length 8-bit character array
Direction	Radio button choices for direction of information: ■ Input —information is directed from the client to the server. ■ Output—information is directed from the server to the client. ■ Both—information is directed from the client to the server, and from the server to the client.
Occurrence(s)	Number of times that an identical field name can be used. If 0, the field name can be used an unlimited number of times. Occurrences are used by JET to build test screens; not to limit information sent or retrieved by BEA Tuxedo.
Screen Information	This button is disabled when the window is launched.
Clear	Clears the fields of the window.
Change	Is disabled while new parameters are added.
Add	Adds new parameters to the service. The parameters are saved when the service is saved.
Back	Returns the user to the previous window.

Adding a Parameter

To add a parameter:

1. Select Field Name to activate the field, and type the field name.

   Note: If the buffer type is FML or VIEW, the field name must match the corresponding parameter field name in FML or VIEW.

2. Select the data type.
3. Specify a direction by selecting the input, output, or both radio buttons.
4. Select the Occurrence text field to activate it, and then enter the number of occurrences.
5. Select Add to append the information. Add does not save the parameter.
6. In the Edit Services window, click Save Service to save the parameter as a part of the service.

   Warning: If you do not click Save Service before you click Back, the parameters are not saved as part of the service.

7. Click Back to return to the Edit Services window.

Selecting CARRAY or STRING as a Parameter Data Type

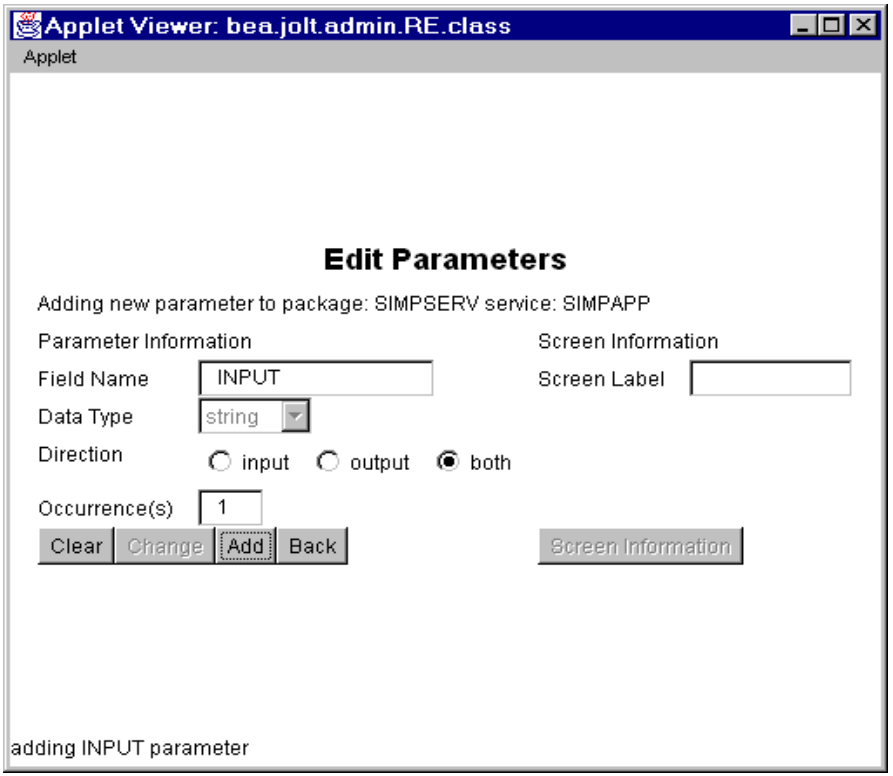
If `CARRAY` or `STRING` is the selected buffer type for a new service, only a carray or string can be added as the data type for the accompanying parameters.

In this case, only one parameter can be added. It is recommended that you use the parameter name `"CARRAY"` for a `CARRAY` buffer type, and the parameter name `"STRING"` for a `STRING` buffer type.

See also “Adding a Service” on page 5-25 and “Selecting CARRAY or STRING as a Service Buffer Type” on page 5-25. For additional information, see Chapter 2, “Invoking BEA Tuxedo Services.”

Figure 5-12 shows an example of the Edit Parameters window with `STRING` as the selected data type for the parameter. The Data Type defaults to `STRING` and does not allow you to modify that particular data type. The Field Name can be any name.

Figure 5-12 Edit Parameters Window: string Data Type



Grouping Services Using the Package Organizer

This topic includes the following sections:

- Sample Package Organizer Window
- Components of the Package Organizer Window
- Grouping Services with the Package Organizer

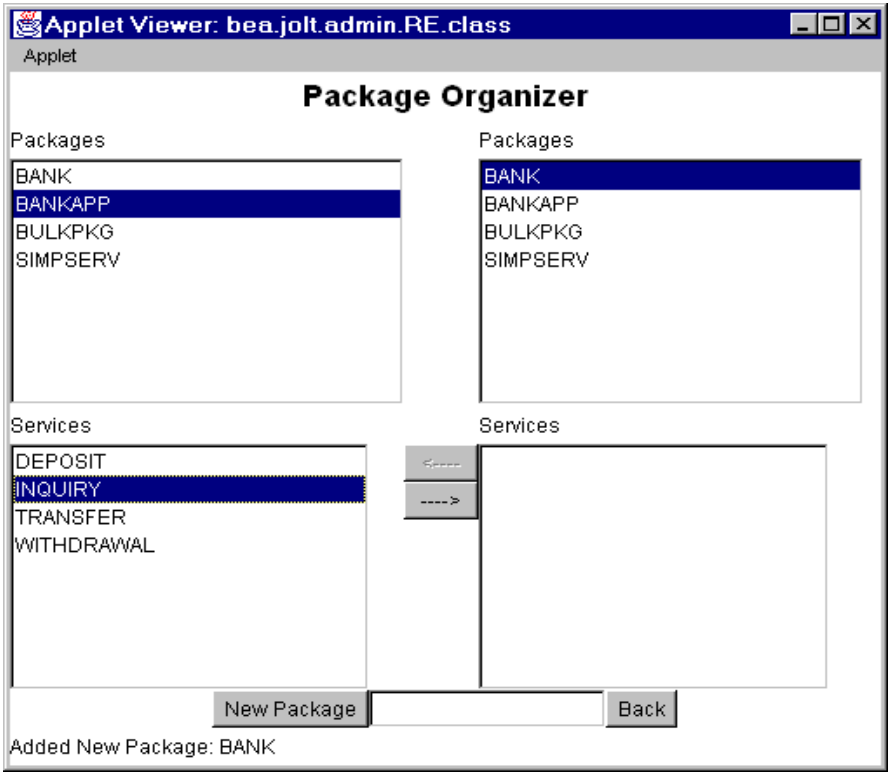
The Package Organizer moves services between packages. You may want to group related services in a package (for example, WITHDRAWAL services that are exported only at a certain time of the day can be grouped together in a package).

Use the Package Organizer arrow buttons to move a service from one package to another. These buttons are useful if you have several services to move between packages. The packages and services display listings to help track a service within a particular package.

Sample Package Organizer Window

Figure 5-13 shows an example of a Package Organizer window with a service selected for transfer to another package.

Figure 5-13 Package Organizer Window



Components of the Package Organizer Window

Table 5-7 describes the components of the Package Organizer window shown in Figure 5-12.

Table 5-7 Package Organizer Window Components

Option	Description
Packages (left display list)	Lists packages containing services in the selected package.

Table 5-7 Package Organizer Window Components (Continued)

Option	Description
Packages (right display list)	Lists packages available as destinations for the selected service.
Services (left display list)	Lists available services for the selected package.
Services (right display list)	Lists available services of the highlighted package that you moved.
Left arrow	Moves services (one service at a time) to the package highlighted on the left.
Right arrow	Moves services (one service at a time) to the package highlighted on the right.
New Package	Adds the name of a new package.
Back	Returns user to the previous window.

Grouping Services with the Package Organizer

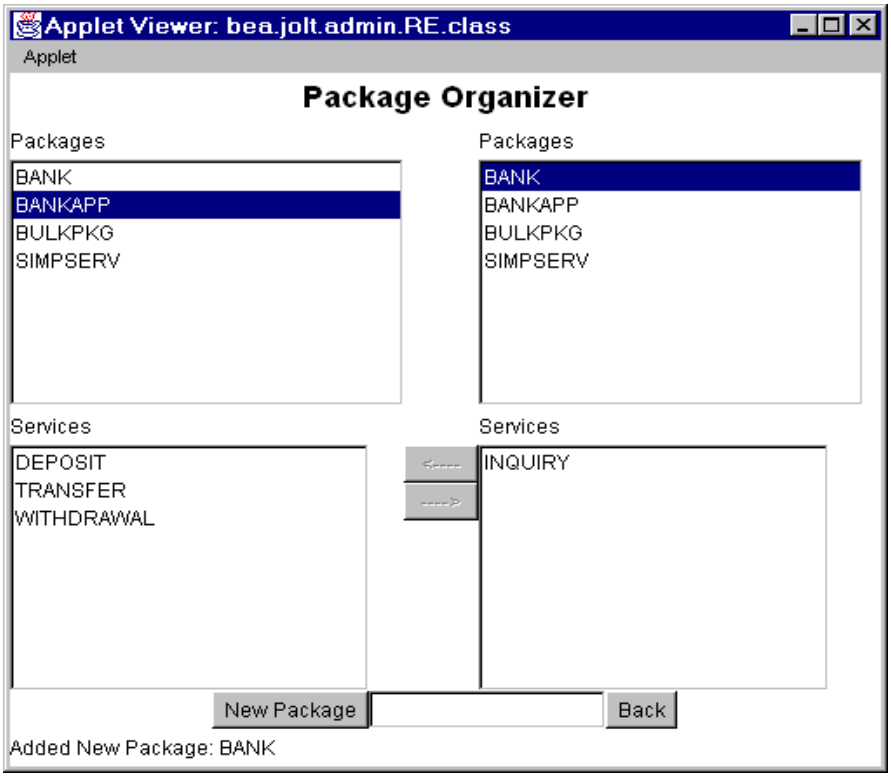
To group services with the Package Organizer:

1. In the Packages window, click Package Organizer.
2. In the Package Organizer window, select the package containing the services to be moved from the Packages left display window.
3. Select the service to be moved from the Services left display window.

   In Figure 5-14, `INQUIRY` is the selected service in the `BANKAPP` package.
4. Select the package to receive the service from the Packages right display window.

   Figure 5-14 shows the selected service, `INQUIRY`, and the selected package, `BANK`, to which the `INQUIRY` service will be moved.

Figure 5-14 Example of a Moved Service



5. To move the service between the packages, select the left arrow **(←)** or right arrow **(→)**.

   These keys are activated only when both packages (left and right are displayed) and a service are selected. The keys are only active in the direction of the package where the service is to be moved. Figure 5-14 shows that the `INQUIRY` service has been moved to the `BANK` package on the right.

   Note: You cannot select the same package in both the left and right display lists.

Modifying Packages, Services, and Parameters

This topic includes the following sections:

- Editing Services
- Editing Parameters
- Deleting Parameters, Services, and Packages

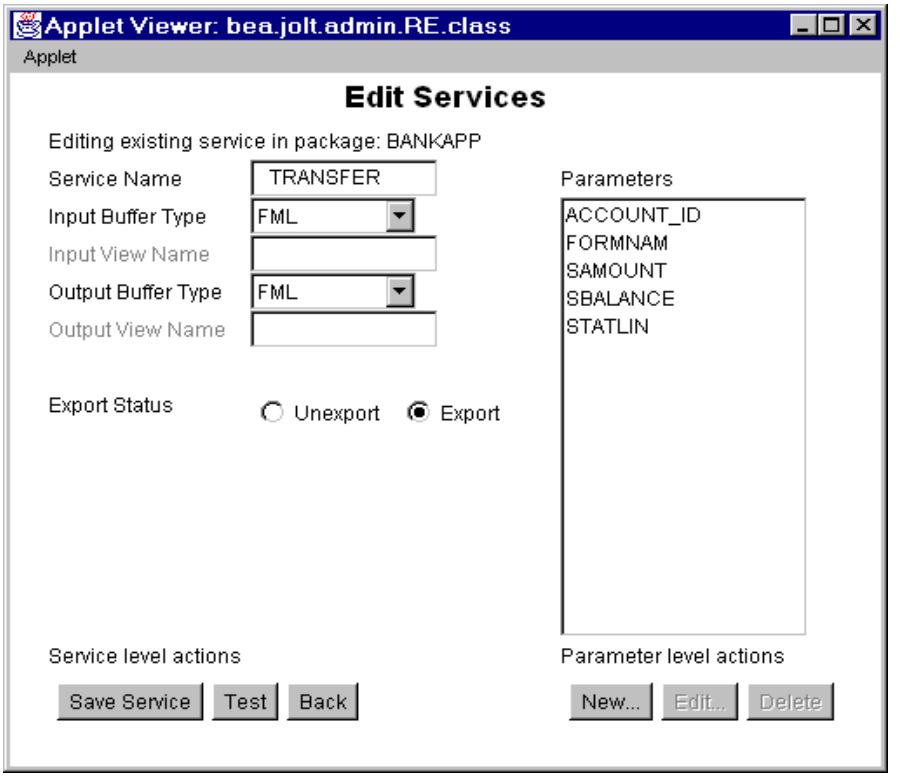
Editing Services

You can edit an existing service name or service information, or access the window to add new parameters to an existing service. For a description of the Edit Services window, see “Options for Adding a Service” on page 5-23.

Sample Edit Services Window

Figure 5-15 shows an example of the Edit Services window.

Figure 5-15 Edit Services Window



Editing a Service

To edit a service:

1. From the Services window, select the package containing the service that requires editing.

   The services available for the selected package are displayed.

2. Select the service to edit.

   The parameters available for the selected service are displayed.

3. Click Edit to display the Edit Services window, as shown in Figure 5-15.

4. Type or select the new information, and click Save Service.

Editing Parameters

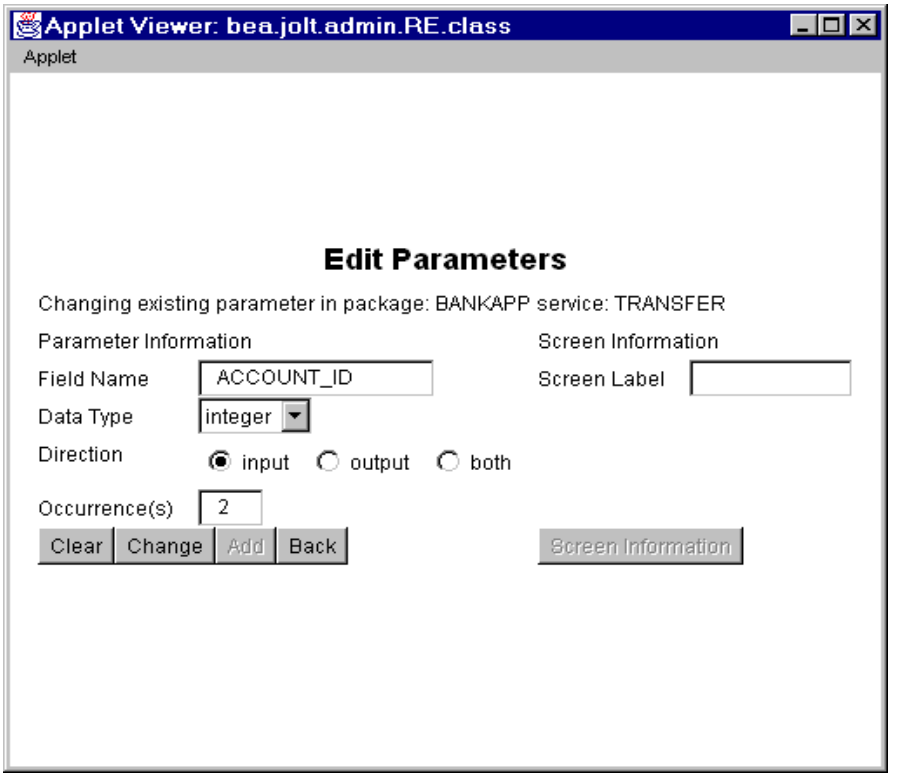
All parameter elements can be changed, including the name of the parameter.

Warning: If you create a new parameter using an existing name, the system overwrites the existing parameter.

Sample Edit Parameters Window

Figure 5-16 shows an example of the Edit Parameters window.

Figure 5-16 Edit Parameters Window



Editing a Parameter

To change a parameter:

1. In the Services window (see “Services Window with Parameters List”), select the package and service that contain the parameter you want to change.
2. Click Edit to display the Edit Services window.
3. Select the Parameter you want to edit from the Parameters display list and click Edit.

   The Edit Parameters Window is displayed as shown in Figure 5-16.
4. Type the new information and click Change.

5. Click Back to return to the previous window.

Deleting Parameters, Services, and Packages

This section describe how to delete a package. Before deleting a package, all the services must be deleted from the package. The Delete option is not enabled until all components of the package or service are deleted.

Warning: The system does not display a prompt to confirm that items are to be deleted. Be certain that the parameter, service, or package is scheduled to be deleted or has been moved to another location before selecting Delete.

Deleting a Parameter

To delete a parameter:

1. In the logon window, click Services to display the Services window.
2. In the Services window, select the package and service that contain the parameter you want to delete.
3. Click Edit to display the Edit Services window.
4. Select the parameter you want to delete from the Parameters display list.
5. Under Parameter Level Actions, click Delete.

Deleting a Service

To delete a service:

Note: Make certain that all parameters within this service are deleted before selecting this option.

1. Select the package containing the service you want to delete.
2. Select the service you want to delete.

   Delete is enabled.
3. Click Delete. The service is deleted.

Deleting a Package

To delete a package:

Note: Make sure all services contained in this package are deleted or moved to another package before selecting this option.

1. In the Jolt Repository Editor Logon window, click Packages to display the Packages window.
2. Select a package.
3. Click Delete.

   The package is deleted.

Making a Service Available to the JET Client

To make a service available to a JET client, you export it. All services in a package must be exported or unexported as a group. A service is made available by using the Export and Unexport radio buttons.

This topic includes the following sections:

- Exporting and Unexporting Services
- Reviewing the Exported and Unexported Status

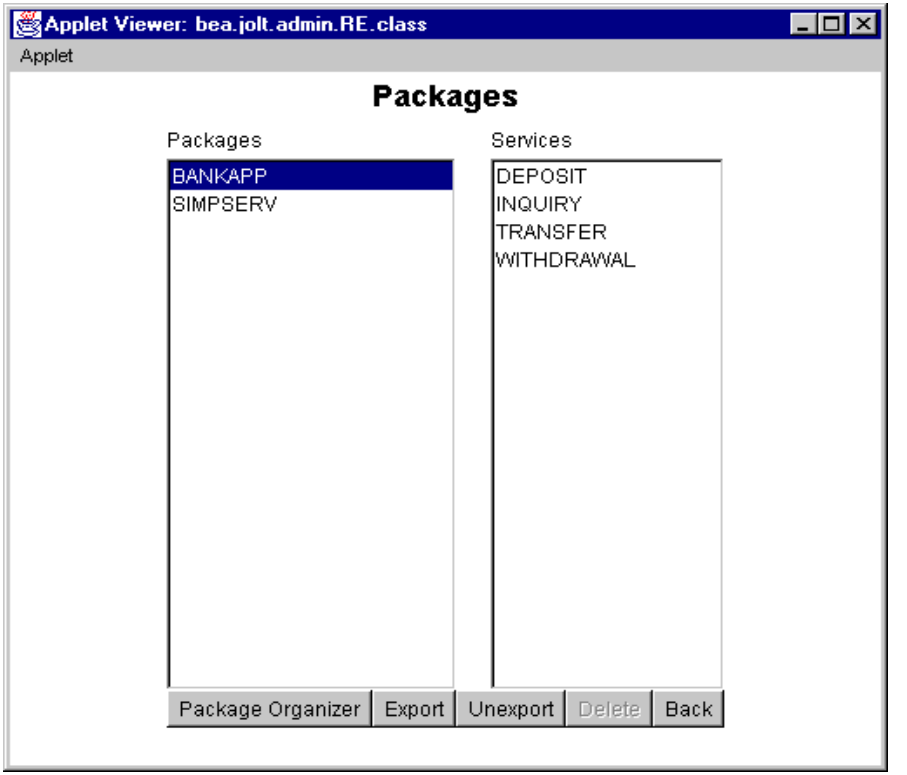
Exporting and Unexporting Services

Determine which services are being made available or unavailable to the JET client. Services are exported to ensure that the JET client can access the most current service definitions from the Jolt server.

Sample Packages Window

Figure 5-17 shows the Packages window, where you can export and unexport services.

Figure 5-17 Packages Window: Export and Unexport Buttons



Exporting or Unexporting a Service

To export or unexport a service:

1. From the Jolt Repository Editor Logon window, select Packages to display the Packages window.
2. Select a package.

   The Export and Unexport buttons are enabled.
3. To make the services in the selected package available, click Export.

   To make the services in the selected package unavailable, select Unexport.

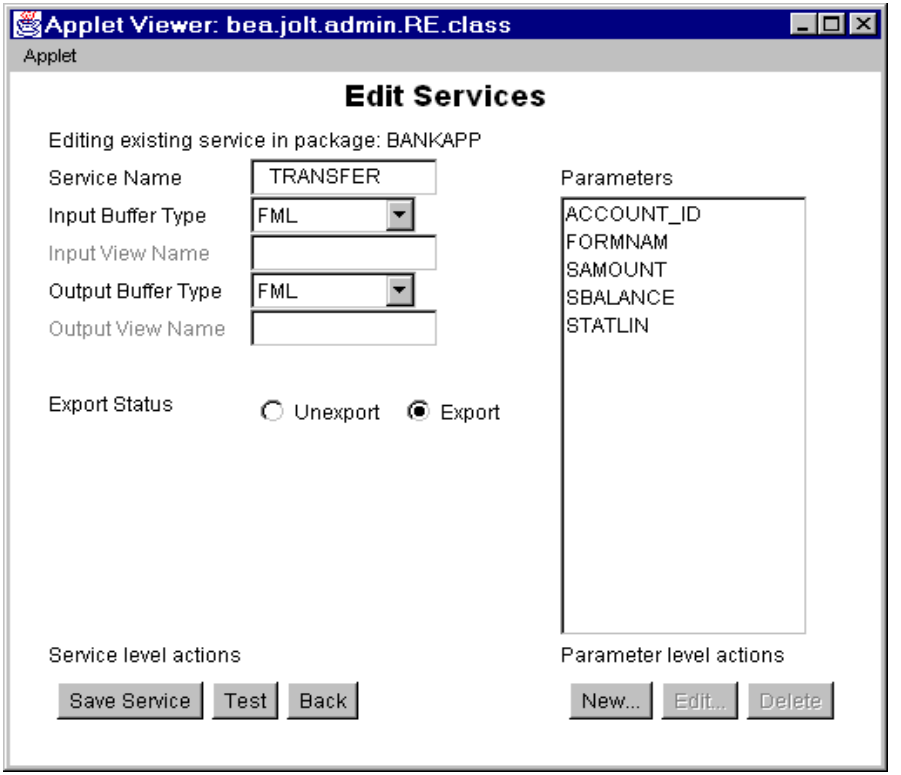
Caution: The system does not display a confirmation message indicating that the service is exported or unexported. See "Reviewing the Exported and Unexported Status" on page 5-42 for additional information.

Reviewing the Exported and Unexported Status

When a service is exported or unexported, you can review its status from the Edit Services window.

Figure 5-18 displays the Export radio button as active, for Export Status; therefore, the current status for the service TRANSFER is exported.

Figure 5-18 Export Status



To review the current exported or unexported status of a service:

1. From the Jolt Repository Editor Logon window, select Services to display the Services window shown in the “Services Window” on page 5-16.
2. Select a package from the Package display list.

   The Services display list of available services for the selected package is displayed.
3. Select the service you want to review.
4. Click Edit.

   The Edit Services window is displayed as shown in Figure 5-15.

   One of the radio buttons (Export or Unexport) next to the Export Status label will be active, indicating the current status of the service.

Testing a Service

This topic includes the following sections:

- Sample Service Test Window
- Components of the Service Test Window
- Testing a Service

Test a service and its parameters before you make them available to JET clients. You can test currently available services without making changes to the services and parameters.

Note: The Jolt Repository Editor allows you to test an existing BEA Tuxedo service with JET, without writing a line of Java code.

An exported or unexported service can be tested; if you need to change a service and its parameters, unexport the service prior to editing.

Sample Service Test Window

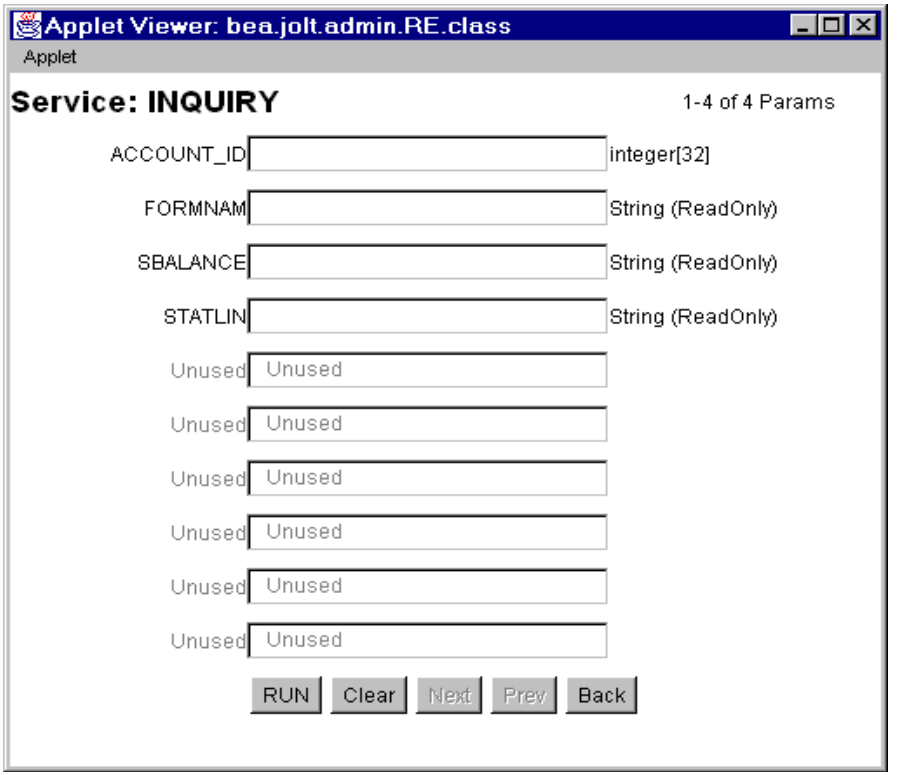
Use the Run button to test the service to ensure that the parameter information is accurate. A service can only be tested when the corresponding BEA Tuxedo server is running for the service being tested.

Although the Test button in the Edit Services window is enabled when parameters are not added to the service, the Service Test window displays `unused` in the parameter fields, and they are disabled. See “Service Test Window” on page 5-45 for an example of unused parameter fields.

Note: The Service Test window displays up to 20 items of any multiple-occurrence parameters. All items that follow the twentieth occurrence of a parameter cannot be tested.

Figure 5-19 shows an example of a Service Test window with both writable and read-only text fields.

Figure 5-19 Service Test Window



Components of the Service Test Window

Table 5-8 describes the components of the Service Test window shown in Figure 5-19.

Note: You can enter a two-digit hexadecimal character (0-9, a-f, A-F) for each byte in the CARRAY data field. For example, the hexadecimal value for 1234 decimal is 0422.

Table 5-8 Service Test Window Components

Option	Description
Service	Displays the name of the tested service (read-only).
Parameters displayed	Tracks the parameters displayed in the window (read-only).
Parameter text fields	The parameter information text entry field. These fields are writable or read-only. Disabled if read-only.
RUN	Runs the test with the data entered.
Clear	Clears the text entry field.
Next	Lists additional parameter fields, if applicable.
Prev	Lists previous parameter fields, if applicable.
Back	Returns to the Edit Services window.

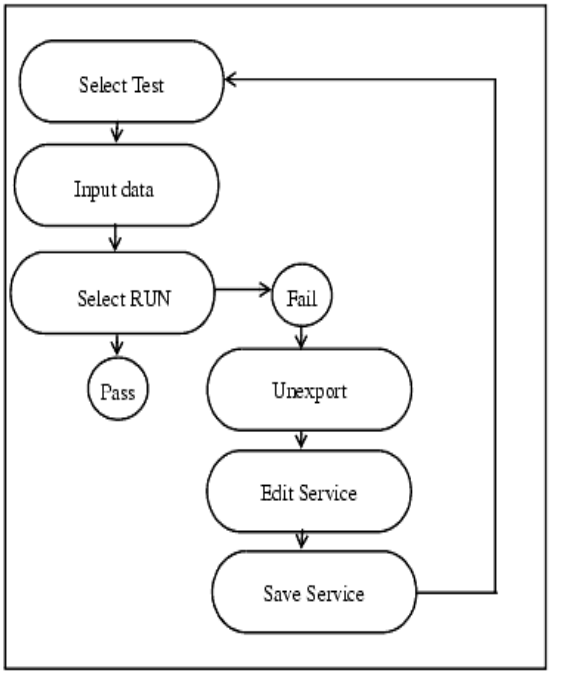
Testing a Service

You can test a service without making changes to the service or its parameters. You can also test a service after editing the service or its parameters.

Test Service Process Flow

Figure 5-20 shows a typical Jolt Repository Editor service flow test.

Figure 5-20 Test Service Flow



Testing a Service

For troubleshooting information, see the first two entries in Table 5-9.

To test a service:

1. Select Services from the Jolt Repository Editor Logon window.

   The Services window is displayed.

2. Select the package and the service to test.
3. Click Edit to access the Edit Services window.
4. Click Test to access the Service Test window.
5. Enter data in the Service test window parameter text fields.
6. Click RUN.

   The status line displays the outcome as follows:

   - If the test passed: "`Run Completed OK`"
   - If the test failed: "`Call Failed`"

     See "Jolt Repository Editor Troubleshooting" on page 5-49 for additional Jolt Repository Editor troubleshooting information.

If edits are required after testing:

1. Return to the Jolt Repository Editor Logon window and click Packages.
2. Select the package with the services to be retested.
3. Click Unexport.
4. Click Back to return to the Jolt Repository Editor Logon window.
5. Click Services to display the Services window.
6. Select the package and the service that requires editing and click Edit.
7. In the Edit Services window, edit the service.
8. Save the service, click Test, and repeat steps 5 and 6 of the "Testing a Service" section.

Troubleshooting

Table 5-9 provides troubleshooting tips if you encounter problems while using the Jolt Repository Editor.

Table 5-9 Jolt Repository Editor Troubleshooting

Problem	Suggested Action(s)
A parameter is incorrect	Edit the service.
The Jolt server is down	Check the server. The BEA Tuxedo service is down. You do not need to edit the service.
You receive any error	Make sure the browser you are running is Java-enabled: ■ For Netscape browsers, make sure that Enable Java and Enable JavaScript are checked under Edit | Preferences | Advanced. Then select Communicator | Tools | Java Console. If the Java Console does not exist on the menu, the browser probably does not support Java. ■ For Internet Explorer, make sure the version is 3.0 (or later). ■ If running Netscape Navigator, check the Java Console for error messages. ■ If running `appletviewer`, check the system console (or the window where you started the `appletviewer`).
You cannot connect to the Jolt server (after entering `Server` and `Port Number`)	Make sure that: ■ Your Server name is correct (and accessible from your machine). Check that the port number is the correct port. A JSL or JRLY must be configured to listen on that port. ■ The Jolt server is up and running. If any authentication is enabled, check that you are entering the correct usernames and passwords. ■ If the applet was loaded through http, the Web server, JRLY and the Jolt server are on the same machine (that is, the server name entered into the Jolt Repository Editor must be the same machine as the one used in the URL to download the applet).

Table 5-9 Jolt Repository Editor Troubleshooting (Continued)

Problem	Suggested Action(s)
You cannot start the Jolt Repository Editor	If you are running the editor in a browser and downloading the Jolt Repository Editor applet through http, make sure that: ■ The browser is Java-enabled. ■ The Web server is running and accessible. ■ The `RE.html` file is available to the Web server. ■ The `RE.html` file contains the correct <codebase> parameter. Codebase identifies where the Jolt class files are located. If running the editor in a browser (or `appletviewer`) and loading the applet from disk, make sure that: ■ The browser is Java-enabled. ■ The `RE.html` file exists and is readable. ■ The `RE.html` file is Java-enabled. ■ The `RE.html` file contains the correct <codebase> parameter (this is where the Jolt class files are installed on the local disk). ■ `CLASSPATH` is set and points to the Jolt class directory.
You cannot display `Packages` or `Services` even though you are sure they exist	■ Make sure that the Jolt Repository server (JREPSVR) is running. ■ Make sure that the JREPSVR can access the Repository file. ■ Make sure of the configuration for JREPSVR: verify CLOPT parameters and verify that `jrep.f16` (FML definition file) is installed and accessible (follow the installation documentation).
You cannot save changes in the Jolt Repository Editor	Check permissions on the Repository file. The file must be writable by the user who starts JREPSVR.
You cannot test services	■ Check that the service is available. ■ Verify the service definition matches the service. ■ If BEA Tuxedo authentication is enabled, check that you have the required permissions to execute the service. ■ Check if the application file (FML or VIEW) is specified correctly in the variables (FIELDTBLS or VIEWFILES) in the ENVFILE. All applications FML field tables or VIEW files must be specified in the FIELDTBLS and VIEWFILES environment variables in the ENVFILE. If these files are not specified, the JSH cannot process data conversion and you receive the following message: `ServiceException: TPEJOLT data conversion failed`. ■ Check the `ULOG` file for any additional diagnostic messages.

Repository Enhancements for Jolt

The Jolt Repository uses the FML32 buffer type, which increases the internal buffer size beyond 64K bytes.

Additionally, the JREPSVR and the Jolt Server (JSH) support the following XATMI buffer types:

- X_OCTET
- X_C_TYPE
- X_COMMON

Index

N

P

R

S

T

U

V

W