



THE ENTERPRISE MIDDLEWARE SOLUTION

# BEA Jolt<sup>tm</sup>



## User's Guide

Jolt 1.1 Release  
Document Edition 1.1  
August 1997

# Copyright

Copyright © 1996, 1997 BEA Systems, Inc. All Rights Reserved.

## Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

## Trademarks or Service Marks

BEA, BEA Connect, BEA Jolt, Distributed Application Framework, and Enterprise Middleware Solutions are trademarks of and are developed and licensed by BEA Systems, Inc., Sunnyvale, California. TUXEDO is a registered trademark of Novell, Inc., exclusively licensed to BEA Systems, Inc.

All other company names may be trademarks of the respective companies with which they are associated.

### Jolt User's Guide

Document Edition	Part Number	Date	Software Version
1.0	801-001103-001	November 1996	Jolt Release 1.0
1.0.1	801-001105-002	February 1997	Jolt Release 1.0.1
1.1	801-001105-003	August 1997	Jolt Release 1.1



---

# Contents

## Preface

Purpose of This Manual.....	xiii
Audience.....	xiii
Jolt Documentation.....	xiii
How This Manual is Organized.....	xiv
Related Manuals .....	xv
Other TUXEDO Resources .....	xv
Document Conventions .....	xvi

## 1. Introducing BEA Jolt

What is BEA Jolt? .....	1-2
Key Features .....	1-3
How it Works .....	1-5
Jolt Servers .....	1-6
Jolt Class Library for Java.....	1-7
Jolt Server and Jolt Client Communication.....	1-8
Jolt Repository.....	1-9
Jolt Internet Relay .....	1-10
How to Jolt your TUXEDO Applications .....	1-11

## 2. Installing Jolt

Installation Requirements .....	2-2
Server Requirements .....	2-2
Jolt Relay Requirements.....	2-3
Client Requirements .....	2-3
BEA Jolt 1.1 Installation .....	2-4
Directory Structure .....	2-4

---

Before You Begin.....	2-6
UNIX System Installation Instructions .....	2-7
UNIX System Installation Script.....	2-8
Windows NT Installation Instructions.....	2-12
Licensing your Jolt Software.....	2-22
Using the Jolt Online Documentation .....	2-26
Getting Started with the Documentation .....	2-27

### **3. Configuring the Jolt System**

Using the Jolt Server.....	3-2
Jolt Internet Relay.....	3-2
Security and Encryption .....	3-3
Starting the Jolt Server .....	3-3
Configuring the Jolt Server .....	3-4
Shutting Down the Jolt Server.....	3-10
Using the Jolt Repository .....	3-10
Configuring the Jolt Repository .....	3-11
Initializing Services Using TUXEDO and the Repository Editor.....	3-13
Event Subscription.....	3-14
Configuration.....	3-14
Jolt Internet Relay.....	3-16
Jolt Relay (JRLY).....	3-18
Jolt Relay Adapter (JRAD) .....	3-20
Using Sample Applications in Jolt Online Resources .....	3-22

### **4. Bulk Loading TUXEDO Services**

Introduction to the Bulk Loader .....	4-1
Getting Started Using the Bulk Loader .....	4-2
Using UNIX.....	4-2
Using Windows NT .....	4-2
Syntax of the Bulk Loader Data Files .....	4-4
Guidelines for Using Keywords .....	4-4
Keyword Order in the Bulk Loader Data File .....	4-5
Using Service-Level Keywords and Values.....	4-6
Using Parameter-Level Keywords and Values.....	4-8

Troubleshooting.....	4-9
Sample Bulk Load Data.....	4-10

## 5. Using the Jolt Repository Editor

Introduction to the Repository Editor.....	5-2
Repository Editor Window.....	5-3
Getting Started.....	5-5
Starting the Repository Editor Using appletviewer.....	5-5
Starting the Repository Editor Using Your Web Browser .....	5-6
Logging on to the Repository Editor.....	5-6
Exiting the Repository Editor.....	5-8
Main Components of the Repository Editor.....	5-10
Repository Editor Flow .....	5-10
What is a Package?.....	5-12
What is a Service?.....	5-15
What is a Parameter?.....	5-17
Setting Up Packages and Services.....	5-19
Saving Your Work .....	5-19
Adding a Package.....	5-19
Adding a Service .....	5-21
Adding a Parameter.....	5-25
Grouping Services Using the Package Organizer .....	5-29
Modifying Packages/Services/Parameters.....	5-33
Editing a Service .....	5-33
Editing a Parameter .....	5-35
Deleting Parameters/Services/Packages.....	5-36
Making a Service Available to the Jolt Client.....	5-38
Exporting/Unexporting Services .....	5-38
Reviewing the Exported/Unexported Status .....	5-40
Testing a Service .....	5-42
Repository Editor Service Test Window.....	5-43
Testing a Service Process Flow.....	5-45
Troubleshooting.....	5-48

---

## 6. Using the Jolt Class Library

Class Library Functionality Overview .....	6-2
Java Applications vs. Java Applets .....	6-2
Jolt Class Library Features .....	6-3
Jolt Client/Server Relationship .....	6-4
Jolt Object Relationships .....	6-7
Jolt Class Functionality .....	6-8
Jolt Class Library Walk-through .....	6-10
Using TUXEDO Buffer Types with Jolt .....	6-14
Using the STRING Buffer Type .....	6-15
Using the CARRAY Buffer Type .....	6-17
Using the VIEW Buffer Type .....	6-19
Using the FML Buffer Type .....	6-19
Multithreaded Applications .....	6-21
Preemptive and Non-preemptive Threads .....	6-21
Using Jolt with Non-Preemptive Threading .....	6-22
Using Threads for Asynchronous Behavior .....	6-23
Using Threads with Jolt .....	6-23
Event Subscription and Notifications .....	6-28
API for Event Subscription .....	6-28
Notification Event Handler .....	6-29
Connection Modes .....	6-30
Notification Data Buffers .....	6-30
TUXEDO Event Subscription .....	6-31
Using the Jolt API to Receive TUXEDO Notifications .....	6-33
Clearing Parameter Values .....	6-35
Reusing Objects .....	6-38
Application Deployment and Localization .....	6-42
Deploying a Jolt Applet .....	6-42
Client Considerations .....	6-43
Web Server Considerations .....	6-43
Localizing a Jolt Applet .....	6-44

---

## 7. Jolt Class Library Reference

Jolt Methods .....	7-2
Methods for Handling Items .....	7-2
JoltSessionAttributes Class .....	7-5
JoltSessionAttributes Constructor .....	7-6
JoltSessionAttributes.....	7-6
JoltSessionAttributes Methods .....	7-7
checkAuthenticationLevel .....	7-7
clear .....	7-8
getBytesDef .....	7-9
getBytesDef.....	7-9
getDoubleDef .....	7-10
getFloatDef.....	7-10
getIntDef .....	7-11
getShortDef .....	7-12
getStringDef .....	7-12
setByte.....	7-13
setBytes .....	7-13
setDouble .....	7-14
setFloat .....	7-14
setInt.....	7-15
setShort.....	7-15
setString .....	7-16
JoltSession Class .....	7-18
JoltSession Constructor .....	7-19
JoltSession.....	7-19
JoltSession Method .....	7-20
endSession.....	7-20
isAlive .....	7-21
onReply .....	7-21
finalize.....	7-22
JoltRemoteService Class .....	7-23
JoltRemoteService Constructor.....	7-24
JoltRemoteService .....	7-24
JoltRemoteService Methods.....	7-24

---

call .....	7-24
JoltRequestMessage Abstract Class .....	7-26
JoltRequestMessage Methods .....	7-28
clear .....	7-31
getApplicationCode .....	7-31
getName .....	7-31
getOccurrenceCount .....	7-32
getByteDef .....	7-32
getBytesDef .....	7-32
getDoubleDef .....	7-33
getFloatDef .....	7-33
getIntDef .....	7-34
getShortDef .....	7-34
getStringDef .....	7-34
getByteItemDef .....	7-35
getBytesItemsDef .....	7-35
getDoubleItemDef .....	7-36
getFloatItemDef .....	7-36
getIntItemDef .....	7-36
getShortItemDef .....	7-37
getStringItemDef .....	7-37
setRequestPriority .....	7-38
setByte .....	7-38
setBytes .....	7-39
setDouble .....	7-39
setFloat .....	7-40
setInt .....	7-40
setShort .....	7-41
setString .....	7-41
setByteItem .....	7-42
setBytesItem .....	7-42
setDoubleItem .....	7-43
setFloatItem .....	7-43
setIntItem .....	7-44
setShortItem .....	7-44



---

setStringItem .....	7-45
addByte .....	7-45
addBytes .....	7-46
addDouble .....	7-46
addFloat .....	7-47
addInt .....	7-47
addShort .....	7-48
addString .....	7-48
delete .....	7-49
deleteItem .....	7-49
JoltTransaction Class .....	7-50
JoltTransaction Constructor .....	7-51
JoltTransaction .....	7-51
JoltTransaction Methods .....	7-52
commit .....	7-52
rollback .....	7-53
JoltEvent Class .....	7-54
JoltEvent Methods .....	7-54
unsubscribe .....	7-54
unsubscribeAll .....	7-55
JoltUserEvent Class .....	7-56
UNSOLMSG .....	7-56
JoltUserEvent Methods .....	7-57
JoltUserEvent .....	7-57
JoltReply Class .....	7-59
JoltReply Methods .....	7-59
getMessage .....	7-59
JoltMessage Class .....	7-60
JoltMessage Methods .....	7-61
getOccurrenceCount .....	7-61
getByteDef .....	7-61
getShortDef .....	7-62
getIntDef .....	7-62
getFloatDef .....	7-63
getDoubleDef .....	7-63

getStringDef .....	7-63
getBytesDef .....	7-64
getByteItemDef .....	7-64
getShortItemDef .....	7-65
getIntItemDef .....	7-65
getFloatItemDef .....	7-66
getDoubleItemDef .....	7-66
getBytesItemDef .....	7-67
getStringItemDef .....	7-67

## A. Jolt Class Library Errors and Exceptions

Jolt Error and Exception Handling .....	A-2
ApplicationException Class .....	A-4
ApplicationException Methods .....	A-5
getMessage Method .....	A-5
getApplicationCode Method .....	A-5
getObject Method .....	A-5
JoltException Class .....	A-6
JoltException Methods .....	A-7
getMessage Method .....	A-7
getErrno Method .....	A-7
getObject Method .....	A-7
EventException Class .....	A-8
MessageException Class .....	A-8
ServiceException Class .....	A-9
SessionException Class .....	A-9
TransactionException Class .....	A-10
TUXEDO Errors .....	A-11

## B. System Messages

Jolt System Messages .....	B-2
Repository Messages .....	B-12
FML Error Messages .....	B-14
Information Messages .....	B-16
Jolt Relay Adapter (JRAD) Messages .....	B-17

---

Jolt Relay (JRLY) Messages .....	B-24
Bulk Loader Utility Messages .....	B-29



---

# Preface



## Purpose of This Manual

This manual describes the BEA Jolt™ product, discusses how to use the Jolt system, and defines messages and terms associated with using the product.

## Audience

This document is intended for system administrators, network administrators, and developers interested in extending secure, scalable transaction-based processing from the enterprise to intranet and Internet wide availability. It assumes a familiarity with BEA TUXEDO and Java programming.

## Jolt Documentation

The Jolt documentation consists of the following documents:

*BEA Jolt User's Guide* (available in both hardcopy and online format)

*BEA Jolt Release Notes* (available in hardcopy format)

---

# How This Manual is Organized

This manual is organized as follows:

Chapter 1, “Introducing BEA Jolt,” describes the Jolt features, architecture, and components.

Chapter 2, “Installing Jolt,” describes how to install the Jolt components.

Chapter 3, “Configuring the Jolt System,” describes security, event notification, the Jolt Relay, and how to configure the Jolt server components.

Chapter 4, “Bulk Loading TUXEDO Services,” describes how to use the Jolt Bulk Loader utility.

Chapter 5, “Using the Jolt Repository Editor,” describes how to add, modify, test, export, and delete TUXEDO service definitions from the Repository based on the information available from the TUXEDO configuration file.

Chapter 6, “Using the Jolt Class Library,” describes how developers use the object-oriented Java language classes for accessing TUXEDO services.

Chapter 7, “Jolt Class Library Reference,” is a reference for Jolt methods and classes.

Appendix A, “Jolt Class Library Errors and Exceptions,” is a resource of Jolt class library errors and exceptions.

Appendix B, “System Messages,” is a resource for Jolt system error messages.

**Note:** BEA TUXEDO and BEA Jolt are trademarked terms. Any occurrence of the terms TUXEDO and Jolt in the document, refers to the BEA TUXEDO and BEA Jolt products.

---

## Related Manuals

*TUXEDO System Reference Manual*

*TUXEDO System Administration Guide*

*TUXEDO System Programmer's Guide, Volumes 1 and 2*

*TUXEDO System Message Manual, Volumes 1 and 2*

## Other TUXEDO Resources

*The TUXEDO System* (Andrade, Carges, Dywer, Felts)

*TUXEDO: An Open Approach to OLTP* (Primatesta)

*Building Client/Server Applications Using TUXEDO* (Hall)

---

# Document Conventions

The following documentation conventions are used throughout this manual:

Item	Convention	Example
Arguments	appear in parentheses and are formatted in a lowercase monospace font. Optional arguments are formatted in italic font. Predefined arguments are formatted in an uppercase font.	<i>(name, 0, value)</i> <code>(ACCTID, 2, 5000)</code>
Caution	Apply to practices that could result in loss of information.	<b>Caution:</b> Be sure to save your information before moving to the next window.
Environment variables	are formatted in an uppercase font.	<code>ENVFILE=\${APPDIR}</code>
Glossary terms	are formatted in italics in the printed copy.	<i>Port</i> is the host name of a Jolt server.
Key names	are presented in boldface type.	Press <b>Enter</b> to continue.
Literals	are formatted in a monospace font.	<code>class extendSample</code>
Notes	highlight procedures and contain information which assist the user in understanding the information contained in this manual.	<b>Note:</b> This feature is available with Jolt.
Programs and applications	are formatted with initial caps.	Use the Repository Editor and the Class Library.



---

User input	are formatted in a monospace font.	Type <code>cd TUXDIR</code>
Warning	applies to practices which could result in loss of productivity or information.	<b>Warning:</b> Be sure to save your information before returning to the previous screen.
Window items	are presented in boldface type. Window items can be window titles, button labels, text edit box names or other parts of the window.	Type your password in the <b>Logon window</b> . Select <b>Export</b> to make the service available to the client.





# 1 Introducing BEA Jolt

BEA Jolt is a Java-based interface to the BEA TUXEDO system that extends TUXEDO services to the Internet. BEA Jolt allows you to build client programs and applets that can remotely invoke existing BEA TUXEDO services allowing application messaging, component management, and distributed transaction processing.

The following BEA Jolt topics are discussed in this chapter:

- ◆ What is BEA Jolt?
- ◆ Key Features
- ◆ How it Works
  - ◆ Jolt Servers
  - ◆ Jolt Class Library for Java
  - ◆ Jolt Server and Jolt Client Communication
  - ◆ Jolt Repository
  - ◆ Jolt Internet Relay
- ◆ How to Jolt your TUXEDO Applications

# What is BEA Jolt?

BEA Jolt is a Java class library and API that provides an interface to BEA TUXEDO from Java clients. The BEA Jolt product consists of several components for creating Java-based client programs that access TUXEDO services and for enabling secure, reliable access to servers inside corporate firewalls. These Jolt components are as follows:

- ◆ **Jolt Servers.** One or more Jolt servers: listen for network connections from clients, translate Jolt messages, multiplex multiple clients into a single process, and submit and retrieve requests to and from TUXEDO based applications running on one or more TUXEDO servers.
- ◆ **Jolt Class Library for Java.** The Jolt class library is a Java package containing the class files which implement the Jolt API. These classes enable Java applications and applets to invoke BEA TUXEDO services. The Jolt class library includes functionality to set, retrieve, manage and invoke communication attributes, notifications, network connections, transactions, and services.
- ◆ **Jolt Repository.** A central Jolt Repository contains definitions of BEA TUXEDO services. These services are used by Jolt at runtime to access TUXEDO services. Using the Repository Editor, you can test new and existing BEA TUXEDO services independently of the client applications. You can export services to a Jolt client application or unexport services by hiding the definitions from the Jolt client.
- ◆ **Jolt Internet Relay.** The Jolt Internet Relay is a component that routes messages from a Jolt client to a Jolt Server Listener (JSL) or Jolt Server Handler (JSH). This eliminates the need for the JSH and TUXEDO to run on the same machine as the Web server. The Jolt Internet Relay consists of the Jolt Relay (JRLY) and the Jolt Relay Adapter (JRAD).

The separation of BEA Jolt into these components permits the transactional and Internet components of client/server applications to be implemented separately with the security and scalability required for large-scale Internet and intranet services.

# Key Features

With BEA Jolt, you can leverage existing TUXEDO services and extend your transaction environment to the corporate intranet or world-wide Internet. The key feature of the Jolt architecture is its simplicity. Using Jolt, you can build, deploy and maintain robust, modular, and scalable electronic commerce systems that operate over the Internet.

BEA Jolt includes the following features.

**Java-based API for Simplified Development.** With its Java-based API, BEA Jolt simplifies application design by providing well-designed object interfaces. Jolt supports the Java JDK 1.02 and is fully compatible with Java threads. Jolt enables Java programmers to build graphical front-ends that use the application and transaction services of TUXEDO without the need to understand detailed transactional semantics or without having to rewrite existing TUXEDO applications.

**Pure Java Client Development.** Using Jolt you can build a pure Java client that runs in any Java-enabled browser. Jolt automatically converts from Java to native BEA TUXEDO data types and buffers and from TUXEDO back to Java. As a pure Java client, your applet or application does not need resident client-side libraries or installation, allowing client applications to be downloaded via the network thereby simplifying software distribution.

**Easy Access to TUXEDO Services via Jolt Repository.** The BEA Jolt Repository facilitates Java application development by managing and presenting BEA TUXEDO services definitions that you can use in your Java client. A Jolt repository bulk loading utility lets you quickly integrate your existing TUXEDO services into the Jolt development environment. Jolt and TUXEDO simplify network and application scalability, while encouraging the reuse of application components.

**GUI-based Maintenance and Distribution of TUXEDO Services.** The Jolt Repository Editor lets you manage BEA TUXEDO service definitions such as service names, inputs and outputs.



The Jolt Repository Editor provides support for different input and output names for services defined in the Jolt Repository.



**Encryption for Secure Transaction Processing.** BEA Jolt allows you to encrypt data transmitted between Jolt clients and the JSL/JSH using a combination of DES and RC4. International packages can use a 40-bit key. United States (U.S.) domestic packages can also use a 128-bit key. Jolt encryption addresses the issue of security that is essential for reliable Internet transaction processing.

**Caution:** Programs using the 128-bit encryption cannot be run outside of the United States. Therefore, clients running 128-bit encryption cannot be outside of the United States. Customers with Intranets extending beyond the United States cannot use this mode of encryption if any internal clients are outside of the United States, without prior documented approval from the United States government.



**Added Security via Internet Relay.** BEA Jolt features an Internet Relay component that allows network administrators to separate their Web Server and TUXEDO application server. Web servers are generally considered insecure as they often exist outside a corporate firewall. The Jolt Internet Relay gives you greater flexibility to locate your BEA TUXEDO server in a secure location or environment on your network, yet still be able to handle transactions from Jolt clients on the Internet.



**Event Subscription Support.** Jolt Event Subscription is used to receive event notifications from either TUXEDO services or other TUXEDO clients. Jolt Event Subscription lets you handle two types of TUXEDO application events:

- ◆ **Unsolicited Event Notifications.** A Jolt client can receive these notifications when a TUXEDO client or service subscribes to unsolicited events and a TUXEDO client issues a broadcast or a directly targeted message.
- ◆ **Brokered Event Notifications.** The Jolt client receives these notifications via the TUXEDO Event Broker. The Jolt client receives these notifications only when it subscribes to an event and any TUXEDO client or server posts an event.

# How it Works

BEA Jolt connects Java clients to applications built using BEA TUXEDO. TUXEDO provides a set of modular services, each offering specific functionality related to the application as a whole. For example, a simple banking application might have services such as INQUIRY, WITHDRAW, TRANSFER, and DEPOSIT. Typically, service requests are implemented in C or COBOL as a sequence of calls to a program library. Accessing a library from a native program means installing the library for the specific combination of CPU and operating system release on the client machine, a situation Java was expressly designed to avoid. The Jolt Server implementation acts as a proxy for the Jolt client, invoking the TUXEDO service on behalf of the client. The BEA Jolt Server accepts requests from the Jolt clients and maps those requests into TUXEDO service requests.

**Figure 1-1 BEA Jolt Architecture**

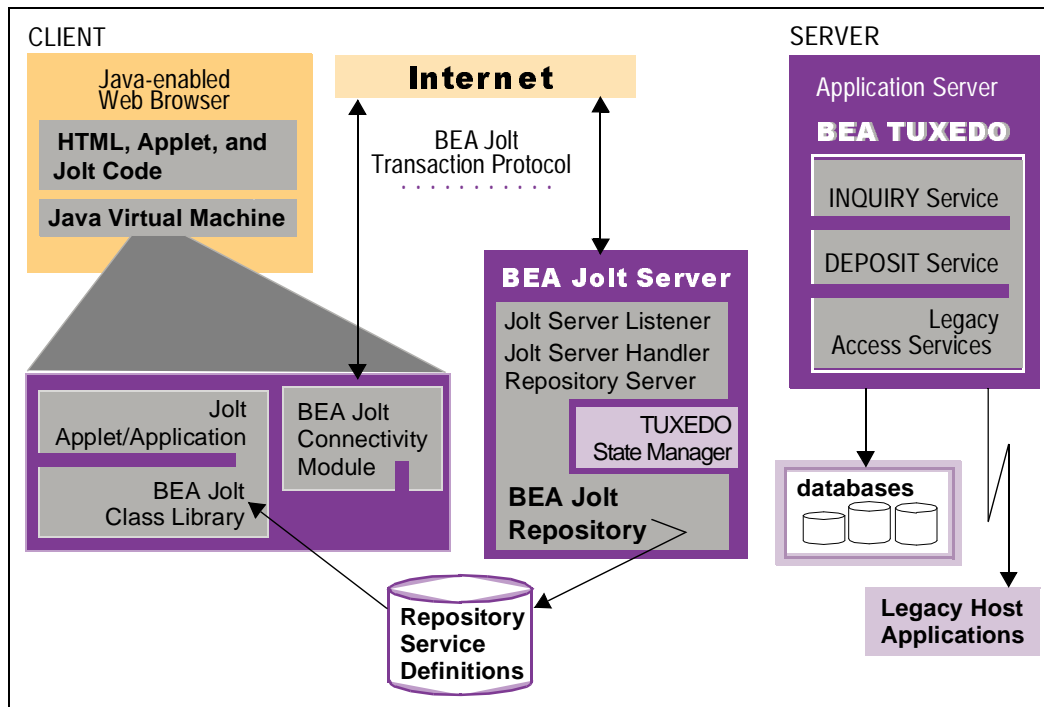


Figure 1-1 illustrates the end-to-end view of the BEA Jolt architecture, as well as related TUXEDO components and their interactions.

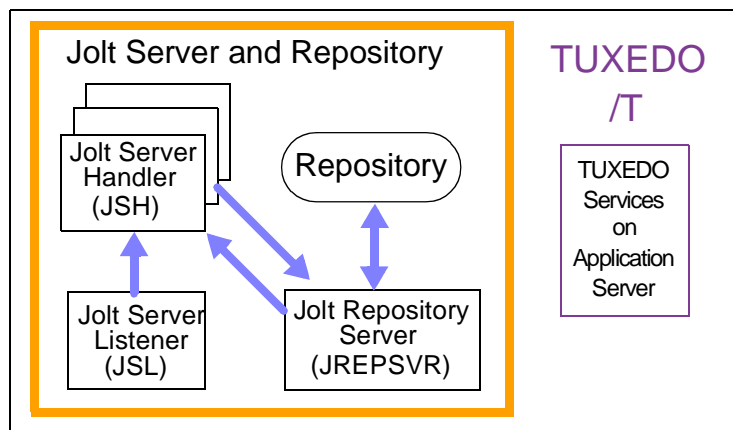
## Jolt Servers

The Jolt Server has several components that act in concert to pass Jolt client transaction processing requests to the TUXEDO application. The components are as follows:

- ◆ **Jolt Server Listener (JSL).** The JSL handles the initial Jolt client connection, and is responsible for assigning a Jolt Server Handler to the Jolt client.
- ◆ **Jolt Server Handler (JSH).** The JSH manages network connectivity, executes service requests on behalf of the client and translates TUXEDO buffer data into the Jolt buffer and vice versa.
- ◆ **Jolt Repository Server (JREPSVR).** The JREPSVR retrieves Jolt service definitions from the Jolt Repository and returns the service definitions to the JSH. The JREPSVR also updates or adds Jolt service definitions.

Figure 1-2 illustrates the Jolt Server and Jolt Repository components.

**Figure 1-2 Jolt Server Components**





## Jolt Class Library for Java

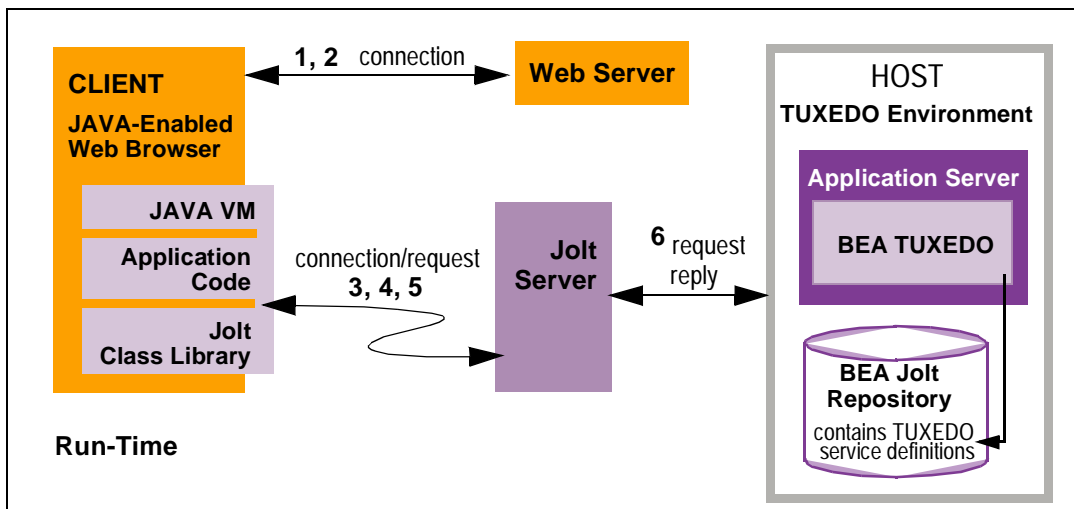
The BEA Jolt Class Library is a set of classes you can use in your Java application or applet to make service requests to TUXEDO from a Java enabled client. These Jolt classes allow you to access TUXEDO transaction services using objects.

When developing a Jolt client application, you only need to know about the classes that Jolt provides and the TUXEDO services that are exported by the Jolt Repository. Jolt hides the underlying application details. Using Jolt and Jolt's 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.

The Jolt API is a Java class library and has the benefits that Java provides: applets are downloaded dynamically and are only resident during runtime. As a result, there is no need for client installation, administration, management, or version control. If services are changed, the client application becomes aware of the changes at the next call to the Jolt Repository.

Figure 1-3 shows the flow of activity from a Jolt client to and from TUXEDO. The call-out numbers correspond to descriptions of the activity in Table 1-1.

**Figure 1-3 Using the Jolt Class Library to access TUXEDO services**



The following table briefly describes the flow of activity involved in using the Jolt Class Library to access TUXEDO services.

**Table 1-1 Using the Jolt Class Library**

Process	Step	Action
<b>Connection</b>  ...  ...	<b>1</b>	A Java enabled Web browser downloads an HTML page using the HTTP protocol.
	<b>2</b>	A Jolt applet is downloaded and executed in the Java Virtual Machine on the client.
	<b>3</b>	The first Java applet task is to open a separate connection to the Jolt Server using a private protocol.
<b>Request</b>  ...	<b>4</b>	The Jolt client now knows the signature of the service (such as, name, parameters, types) and can build a service request object based on Jolt class definitions, and make a method call.
	<b>5</b>	The request is sent to the Jolt Server, which translates the Java based request into TUXEDO requests and forwards the request to the TUXEDO environment.
<b>Reply</b>	<b>6</b>	The TUXEDO system processes the request and returns the information to the Jolt Server, which translates it back to the Java applet.

## Jolt Server and Jolt Client Communication

The Jolt system handles all communication between the Jolt Server and the Jolt client using the BEA Jolt Transaction Protocol. The communication process between the Jolt Server and the Jolt client applet or applications functions as follows:

1. TUXEDO service requests and associated parameters are packaged into a message buffer and delivered over the network to the Jolt Server.
2. The Jolt Server unpacks the data from the message, and performs any data conversions necessary, such as numeric format conversions or character set conversions.

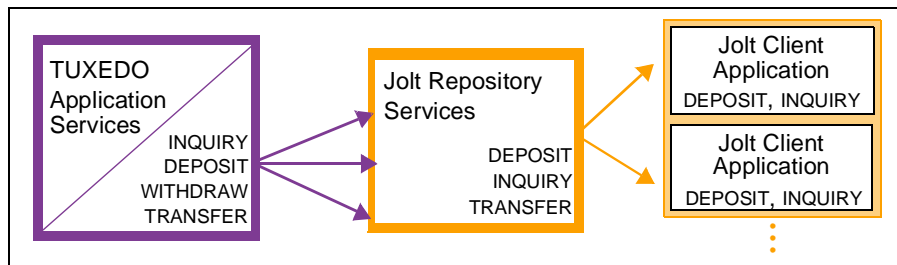
3. The Jolt Server makes the appropriate service request to the application service requested by the Jolt client.
4. Once a service request enters the BEA TUXEDO system, it is executed in exactly the same manner as requests issued by any other TUXEDO client.
5. The results are then returned to the BEA Jolt Server, that packages the results and any error information into a message that is sent to the Jolt client applet.
6. The Jolt client then maps the contents of the message into the various Jolt client interface objects, completing the request.

## **Jolt Repository**

The Jolt Repository is a database where TUXEDO services are defined, such as name, number, type, parameter size, and permissions. The Repository functions as a central database of definitions for TUXEDO services and permits new and existing TUXEDO services to be made available to Jolt client applications. A TUXEDO application can have many services or service definitions such as `ADD_CUSTOMER`, `GET_ACCOUNTBALANCE`, `CHANGE_LOCATION`, `GET_STATUS`. All or only a few of these definitions may be exported to the Jolt Repository. Within the Jolt Repository, the developer or system administrator can export these services to the Jolt client application.

All Repository services that are exported to one client are exported to all clients. TUXEDO handles the cases where subsets of services may be needed for one client and not others. Figure 1-4 illustrates how the Jolt Repository brokers TUXEDO services to multiple Jolt client applications. The diagram shows four TUXEDO services, however the `WITHDRAW` service is not defined in the Repository and the `TRANSFER` service is defined but not exported.

**Figure 1-4 Distributing TUXEDO Services via Jolt**



## Jolt Repository Editor

The Jolt Repository Editor is a Java-based GUI administration tool that gives the application administrator access to individual BEA TUXEDO services. With the Jolt Repository Editor you can define, test, and export services to Jolt clients.

The Jolt Repository Editor enables you to extend and distribute TUXEDO services to Jolt clients without having to modify many lines of code in widely distributed client applications. With the Jolt Repository Editor, you can modify parameters for TUXEDO services, logically group TUXEDO services into packages, and remove services from created packages. You can also make the services available to Web browser-based Jolt applets or Jolt applications by exporting the services.

**Note:** The Jolt Repository Editor only controls services for Jolt client applications. It cannot be used to make changes to the TUXEDO application.

## Jolt Internet Relay

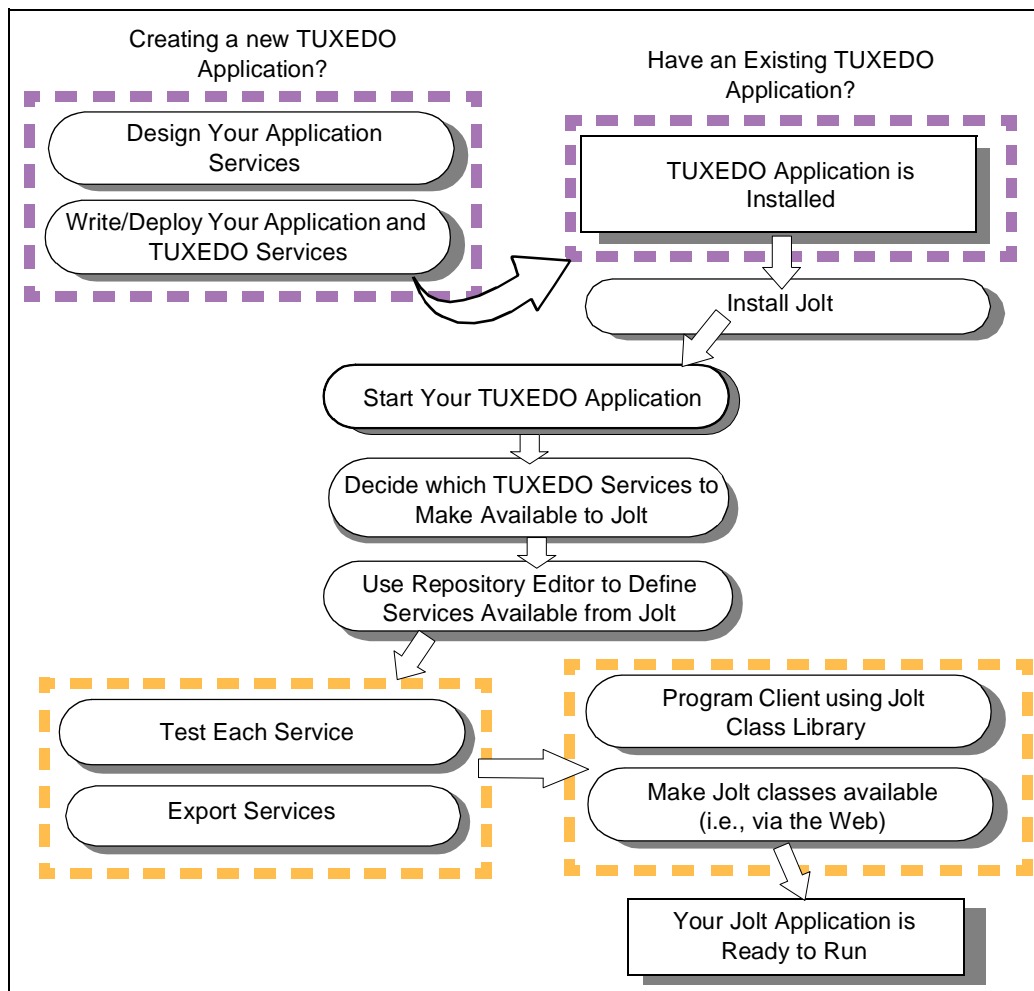
The Jolt Internet Relay is a component that routes messages from a Jolt client to a JSL or JSH. The Jolt Internet Relay consists of the **Jolt Relay (JRLY)** and the **Jolt Relay Adapter (JRAD)**. The Jolt Relay is not a TUXEDO client or server. JRLY is a stand-alone software component that routes Jolt messages to the Jolt Relay Adapter. Requiring only minimal configuration to allow it to work with Jolt clients, the Jolt Relay eliminates the need for TUXEDO to run on the same machine as the Web server.

The JRAD is a TUXEDO application server, but does not include any TUXEDO services. It requires command line arguments to allow it to work with the JSH and TUXEDO. JRAD receives client requests from JRLY, and forwards the request to the appropriate server. Replies from the server are forwarded back to JRAD, which sends the response and back to the requesting client. A single Jolt Internet Relay (JRLY/JRAD pair) handles multiple clients concurrently.

# How to Jolt your TUXEDO Applications

Figure 1-5 illustrates how to Jolt-enable your new and existing TUXEDO-based applications. The process for creating Jolt clients is described in the following steps.

**Figure 1-5 Creating a Jolt Application**



The following steps show just how quickly and easily Jolt clients can be created and deployed.

1. Begin the process with a TUXEDO application.

For information about installing TUXEDO and creating a TUXEDO application, refer to the *TUXEDO System 6 documentation set*.

2. Install Jolt.

For information about installing Jolt components, refer to “Installing Jolt” in Chapter 2.

3. Configure and define services using the Jolt Repository Editor.

4. For information regarding configuring the Jolt Repository Editor and making TUXEDO services available to Jolt, refer to:

- ◆ “Using the Jolt Repository Editor” in Chapter 5
- ◆ “System Messages” in Appendix B

5. Create a client application using the Jolt Class Library.

The following documentation shows you how to program your client application using the Jolt Class Library:

- ◆ “Using the Jolt Class Library” in Chapter 6
- ◆ “Jolt Class Library Reference” in Chapter 7
- ◆ “Jolt Class Library Errors and Exceptions” in Appendix A

6. Run the Jolt-based client applet or application.

Refer to “Using the Jolt Class Library” in Chapter 6 to assist you in installing a Jolt Class Library on a Web Server.



# 2 Installing Jolt

This chapter explains how to install the Jolt 1.1 software and its online documentation. Readers of this chapter are assumed to be system administrators and/or application developers who have experience with the operating platforms on which they are going to install BEA TUXEDO and Jolt software.

This chapter includes the following sections:

- ◆ Installation Requirements
- ◆ BEA Jolt 1.1 Installation
- ◆ Using the Jolt Online Documentation



# Installation Requirements

The following hardware and software components are required before installing BEA Jolt.

## Server Requirements

- ◆ CD-ROM access
- ◆ 500K of disk storage

**Note:** Jolt 1.1 server platform support is dependent on the TUXEDO version support. For example, Jolt 1.1 will only run on Solaris 2.4 and 2.5 if TUXEDO 6.1 or 6.2 is running on the same machine as the Jolt 1.1 server.

- ◆ DEC UNIX 4.0
- ◆ DEC Alpha NT 4.0
- ◆ Hewlett-Packard HP9000 with HP-UX 10.10, 10.20
- ◆ IBM RS/6000 with AIX 4.1.4, 4.2
- ◆ Intel with Windows NT 3.51 or NT 4.0 (80486 processor or later) (TUXEDO 6.3 is available only on NT 4.0)
- ◆ Sequent Dynix 4.2
- ◆ SGI Irix 6.2
- ◆ Sun SPARC with Solaris 2.4, 2.5, 2.5.1 (TUXEDO 6.3 supports 2.5.1 only)
- ◆ Unixware 2.1

## Jolt Relay Requirements

The Jolt Relay supports the following platforms running on the Web server:

- ◆ Hewlett-Packard HP9000 with HP-UX 10.10, 10.20
- ◆ Intel with Windows NT 4.0
- ◆ Sun SPARC with Solaris 2.5

## Client Requirements

- ◆ 700K of disk storage (for application development and Web server)
- ◆ Java-enabled browser (see the *Jolt Release Notes* for the approved browsers) or Java virtual machine
- ◆ Java Developer's Kit (JDK) 1.0.2 (for application development only)  
(<http://java.sun.com:80/java.sun.com/products/JDK/index.html>)
  - ◆ JDK 1.0.2 on Solaris
  - ◆ JDK 1.0.2 on Solaris (appletviewer)
- ◆ HTML browser for the online documentation (Netscape Navigator 2.02, or Microsoft Internet Explorer 3.0 or later are recommended)

# **BEA Jolt 1.1 Installation**

You can install the Jolt 1.1 package from a CD-ROM for UNIX and Windows NT platforms. Your CD-ROM contains all of the necessary files for installing and running your Jolt product, including the Jolt Internet Relay. The Jolt Relay Front-End is installed on the Web server machine. For the Jolt 1.1 release, this machine may be different from the TUXEDO/Jolt machine. You may be required to run the installation program a second time, using the machine that will run the Jolt Relay Front-End.

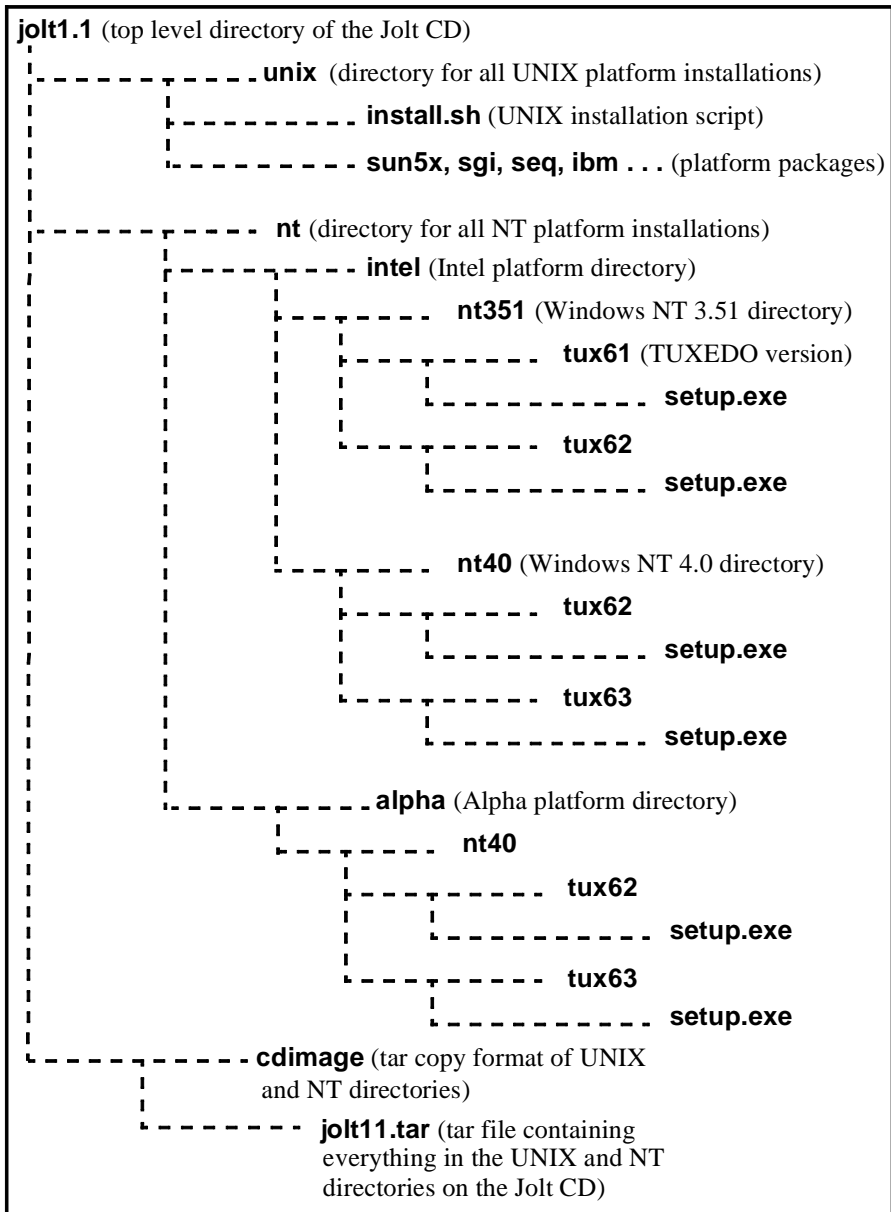
Online documentation, in the form of HTML files, is also available on the CD-ROM.

The CD-ROM contains an installation script for UNIX systems and a separate GUI-based installer for Windows NT users. To install Jolt 1.1, follow the instructions for your respective platform.

## **Directory Structure**

Figure 2-1 shows the directory structure for UNIX and NT systems.

Figure 2-1 Sample Directory Structure



# Before You Begin

Before installing:

- ◆ Verify the location of the TUXEDO directory where the Jolt server is going to be installed.
- ◆ Determine the location of the documentation directory where the Jolt documentation is going to be installed.
- ◆ Verify the Web server location where the Jolt client components are going to be installed.
- ◆ Verify the user ID and group ID assigned to Jolt server files.
- ◆ Verify the user ID and group ID to be assigned to Jolt client files.
- ◆ Review the *Jolt Release Notes* and the Jolt Home Page for any new information.

# UNIX System Installation Instructions

The Jolt 1.1 installation shell script for UNIX systems includes all components necessary for installing the Jolt 1.1 system, the Jolt Repository, the Jolt Server, and the Jolt Class Library code. Refer to Figure 2-1 for an example of the Jolt directory structure.

When installing, ensure that all necessary hardware and software has been installed.

1. Log in as a user who has write permission in the TUXEDO directory.
2. Insert the CD-ROM in the CD-ROM drive. If you are running on Solaris and the daemon `/usr/sbin/vold` is running, the CD-ROM should be automatically mounted in the `/cdrom/JOLT` directory.

```
cd /cdrom/jolt/unix
```

If you are not running on Solaris or `vold` is not running, consult your UNIX administration documentation to mount the CD-ROM.

3. Type `ls`

The directory contents should look similar to the following sample. If not, verify that you are installing the correct CD-ROM.

```
alpha/      hp/          ibm/
install.sh
seq/        sgi/         sun5x/   uw/
```

4. Type

```
sh install.sh
```

5. Press **Enter**.

This invokes the Jolt installation script. The step-by-step install screens are described in the following section.

## UNIX System Installation Script

The UNIX system installation script provides a set of step-by-step instructions to help you quickly install your Jolt product. This script lets you specify your platform, operating system, and other installation details. The installation script prompts you through the entire installation process. You can cancel the installation at any time by pressing **CTRL-C** simultaneously.

**Note:** The script used to show the UNIX installation is taken from Jolt 1.1 for TUXEDO 6.1/6.2. There are variations of the UNIX installation script for Jolt 1.1 for TUXEDO 6.3.

1. Type the number that corresponds to the name of the operating system you are using (for example, if using SPARC Solaris 2.5.1 for TUXEDO 6.2, type 22). Press **Enter**.

```
01) alpha/dux40/6.2  02) alpha/dux40/6.3  03) hp/hp1010/6.1
04) hp/hp1010/6.2  05) hp/hp1010/6.3  06) hp/hp1020/6.1
07) hp/hp1020/6.2  08) hp/hp1020/6.3  09) ibm/aix414/6.1
10) ibm/aix414/6.2  11) ibm/aix414/6.3  12) ibm/aix42/6.1
13) ibm/aix42/6.2  14) ibm/aix42/6.3  15) seq/dynix42/6.1
16) seq/dynix42/6.2  17) seq/dynix42/6.3  18) sgi/irix62/6.1
19) sgi/irix62/6.2  20) sgi/irix62/6.3  21) sun5x/sol24/6.1
22) sun5x/sol24/6.2  23) sun5x/sol25/6.1  24) sun5x/sol25/6.2
25) sun5x/sol251/6.1  26) sun5x/sol251/6.2  27) sun5x/sol251/6.3
28) uw/uw21/6.1      29) uw/uw21/6.2      30) uw/uw21/6.3
Install which platform's files? [01-30, q to quit, l for list]:
22
** You have chosen to install from sun5x/sol24/6.2 **
```

2. You are prompted to review the directory containing the BEA Jolt system. If correct, type y for “yes,” or n for “no” or q to “quit.” Press **Enter**.

```
BEA Jolt Release 1.1
This directory contains the BEA Jolt System for
Solaris 2.4 on SPARC
Is this correct? [y,n,q]: y
To terminate the installation at any time
press the interrupt key,
typically <del>, <break>, or <ctrl+c>.
```

3. Type `BEA Jolt` to install the BEA Jolt package. Press **Enter**.

The following packages are available:

```
1      jolt          BEA Jolt
Select the package(s) you wish to install (or 'all' to install
all packages) (default: all) [?,??,q]: 1
BEA Jolt
(sparc) Release 1.1
Copyright (c) 1997 BEA Systems, Inc.
Portions * Copyright 1986-1997 RSA Data Security, Inc.
All Rights Reserved.
Distributed under license by BEA Systems, Inc.
TUXEDO is a registered trademark.
BEA Jolt is a trademark of BEA Systems, Inc.
```

4. Type the number of the installation option you prefer. The Jolt Server is installed in an existing TUXEDO directory. You must install TUXEDO prior to installing Jolt.

The following installation options are available:

```
1      all           Install the full Jolt System
2      server        Install the server only
3      client        Install the client only
4      rad           Install the relay back-end only
5      doc           Install the documentation
Select an option (default: all) [?,??,q]: 1
Note that the jolt server will be installed into an existing
TUXEDO directory.  You MUST have previously installed TUXEDO
version 6.1, 6.2, or 6.3 to attempt this installation.
Base directory of existing TUXEDO installation [?,q]:
/usr/jolt/T6.2
Determining if sufficient space is available ...
1118 blocks are required
167860 blocks are available to /usr/jolt/T6.2
Using /usr/jolt/T6.2 as the TUXEDO base directory
The client software should be installed either on your web
server machine, or a machine easily accessible to your web
server machine, as the class files must be downloaded.
JOLTDIR below refers to the directory in which your java
related files are stored.  It is the directory which contains
the directory 'classes', not the classes directory itself!
JOLTDIR (default: /usr/jolt/T6.2/udataobj/jolt) [?,q]:
```



```
Determining if sufficient space is available ...
1118 blocks are required
167860 blocks are available to /usr/jolt/T6.2/udataobj/jolt
Using /usr/jolt/T6.2/udataobj/jolt as the Jolt client tree
Unloading /host/sansei/cdrom/sun5x/sol24/6.2/jolt/joltall.Z ...
bin/JREPSVR
bin/JSL
bin/JSH
bin/joutil
bin/JRAD
udataobj/jrep.fl6
udataobj/jwsladmin.f32
udataobj/jrepository
udataobj/jolt/client/Atm.html
udataobj/jolt/client/RE.html
udataobj/jolt/client/jolt.zip
udataobj/jolt/client/audio/dot.au
udataobj/jolt/client/audio/ring.au
udataobj/jolt/client/audio/splat.au
udataobj/jolt/client/images/beaLogo.gif
.
.
.

udataobj/jolt/relay/jrly
udataobj/jolt/relay/jrly.config
locale/C/JOLT_CAT
locale/C/JOLT.text
locale/C/JRAD_CAT
locale/C/JRAD.text
lib/libjconv.so
lib/libjnw.so
include/jotypes.h
2280 blocks
... finished
```

5. Type your Jolt serial number and press **Enter**. Type your Jolt license token number and press **Enter**. The script continues with the installation process until the status message, “Installation of BEA Jolt was successful,” displays.

```
Serial number [?,q]: <enter BEA-provided serial number>
License token [?,q]: <enter BEA-provided license token>
LICUTIL is /usr/jolt/T6.2/bin/joutil
users=0
pbtype=SDK
expdate=9801
serial=<BEA serial number displays>
token=<BEA license token displays>
lictype=a
Activating the license for software in /usr/jolt/T6.2 ...
... finished
Changing file permissions...
... finished
Installation of BEA Jolt was successful
```

6. The script returns to the installation prompt. Type **q** to quit.

```
The following packages are available:
  1      jolt          BEA Jolt
Select the package(s) you wish to install (or 'all' to install
all packages) (default: all) [?,??,q]: q
Please don't forget to fill out and send in your registration
card
```

7. When the installation is complete, unmount the CD-ROM.

The installation is now complete.

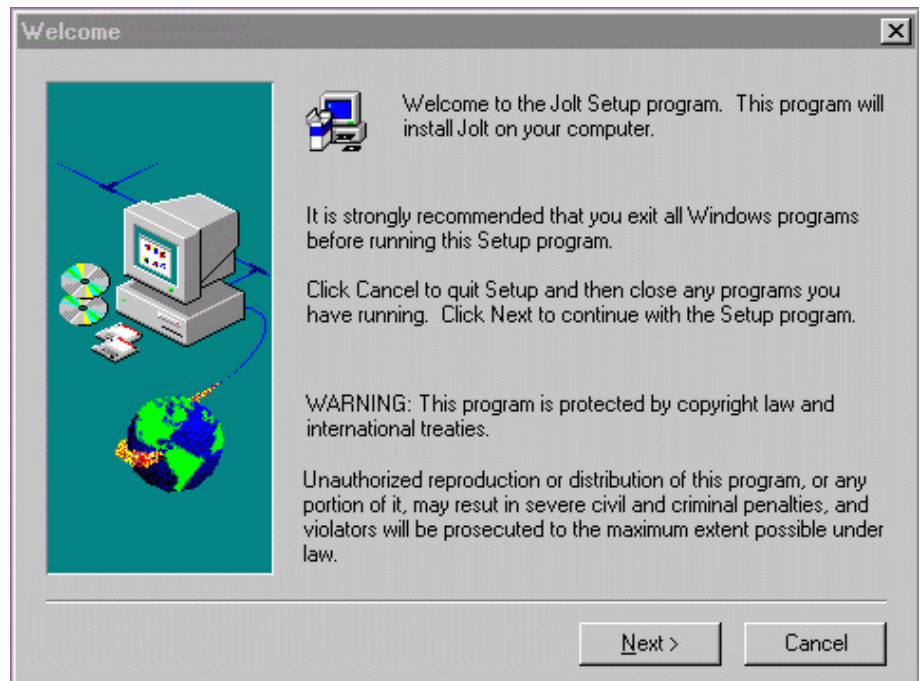
## Windows NT Installation Instructions

The Jolt NT Installer provides a set of step-by-step installation windows to help you quickly install your Jolt product. These windows automate the details of your installation process and prompt you through the entire installation. You can cancel the installation at any time.

If you are installing the Jolt package from a CD-ROM, use Windows Explorer or a similar utility. Select the `setup.exe` program in the NT directory that matches your platform and TUXEDO version when you insert the CD-ROM. Refer to Figure 2-1 for additional information on the directory structure.

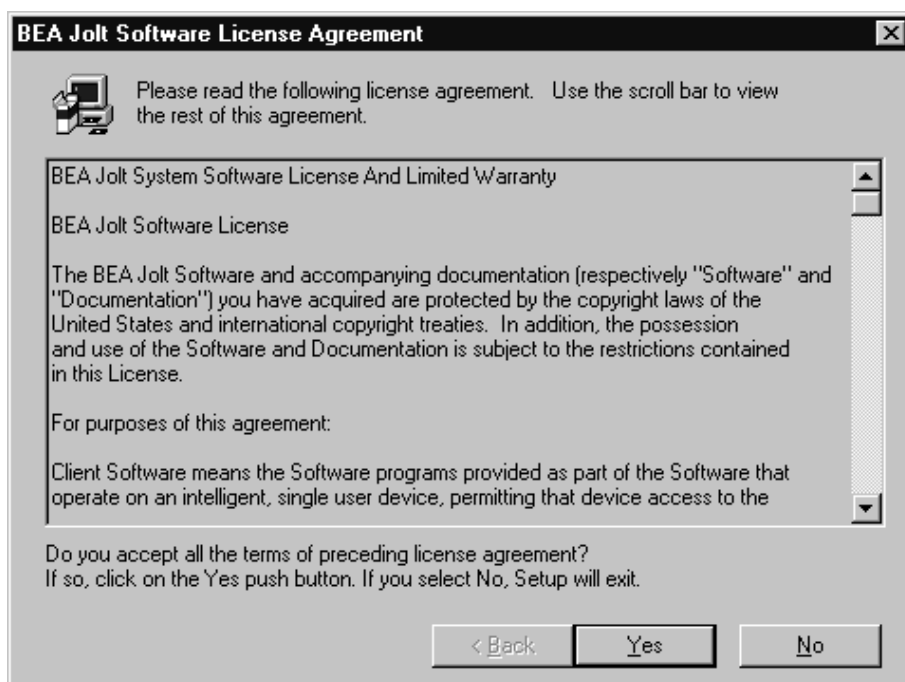
1. When you see the Welcome window shown in Figure 2-2, select the **Next** button to proceed with the installation.

**Figure 2-2 Jolt Welcome Window**



2. Use the scroll bar or the **Page Down** key to read the Software License Agreement. To continue with the Jolt installation, you must accept the terms of the license agreement. If you accept the terms, select **Yes** to continue with the installation. If you do not accept the terms, select **No** and the installation stops.

**Figure 2-3 Software License Agreement**



3. At the User Registration window, type your name and company. Use the **Tab** key to navigate through the text fields. Select **Next** after you enter the information.

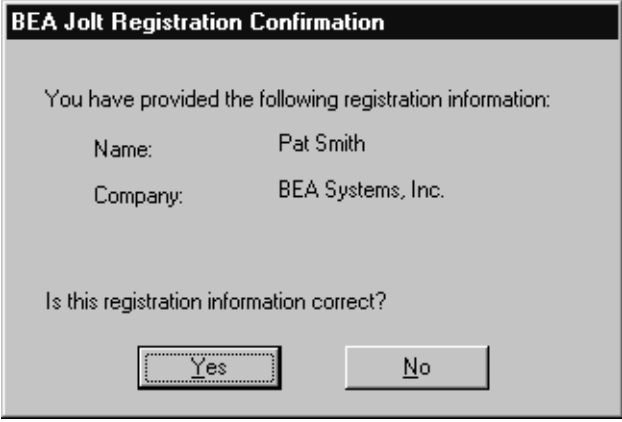
**Figure 2-4 User Registration Window**



The image shows a Windows-style dialog box titled "BEA Jolt User Registration". On the left side, there is a vertical rectangular area containing a graphic of a computer monitor, a CD-ROM, and a globe with a network connection line. To the right of this graphic, the text "Please Register your product now." is displayed. Below this text are two text input fields. The first field is labeled "Name:" and contains the text "Pat Smith". The second field is labeled "Company:" and contains the text "BEA Systems, Inc.". At the bottom right of the dialog box, there are three buttons: "< Back", "Next >", and "Cancel". The "Next >" button is highlighted with a darker border.

4. At the Registration Confirmation window, review the information. If the information is correct, select **Yes** to continue with the installation. If the information is not correct, select **No** to return to the User Registration window and change the registration information.

**Figure 2-5 Registration Confirmation**



The image shows a Windows-style dialog box titled "BEA Jolt Registration Confirmation". The background is light gray. The title bar is black with white text. The main area contains the text "You have provided the following registration information:" followed by two lines of information: "Name: Pat Smith" and "Company: BEA Systems, Inc.". Below this, the question "Is this registration information correct?" is displayed. At the bottom, there are two buttons: "Yes" and "No". The "Yes" button is highlighted with a dashed border, indicating it is the default or selected option.

**BEA Jolt Registration Confirmation**

You have provided the following registration information:

Name: Pat Smith

Company: BEA Systems, Inc.

Is this registration information correct?

- At the Installation Selection window, select the modules to install (in this case, **Jolt Server**, **Jolt Client**, and **Jolt Documentation**).

This window also allow you to choose a destination directory. To do this, select **Browse** and choose a distention path. To continue with the installation, select **Next**.

**Figure 2-6 Installation Selection Window**



**Table 2-1 Setup Type Window**

Setup Type...	Installs...
Jolt Server	Jolt server only (review license)
Jolt Client	Jolt client only
Jolt Relay Back-End	Jolt Internet Relay on the back-end server only
Jolt Relay Front-End	Jolt Internet Relay on the front-end server only
Jolt Documentation	HTML-based documentation

- a. The selection works as a toggle. With the mouse, click the left mouse button once to select and once to deselect. Press the **Tab** key once to highlight and select. Press it again to highlight and select the next item in the list.
- b. To make your selection, click on the space to the left of the text representing your choice. The window displays a checkmark to the left of your selection.  
To deselect a component, click on the checkmark to the left of your choice and the checkmark is removed.
- c. To choose a different destination directory, select **Browse** and choose a destination path.

**Note:** The destination path must be the TUXEDO directory.



6. Select **Disk Space** to check the available and required disk space on a particular drive. Figure 2-7 shows the available disk space. Select **OK** to continue. Select **Cancel** to return to the Installation Selection window.

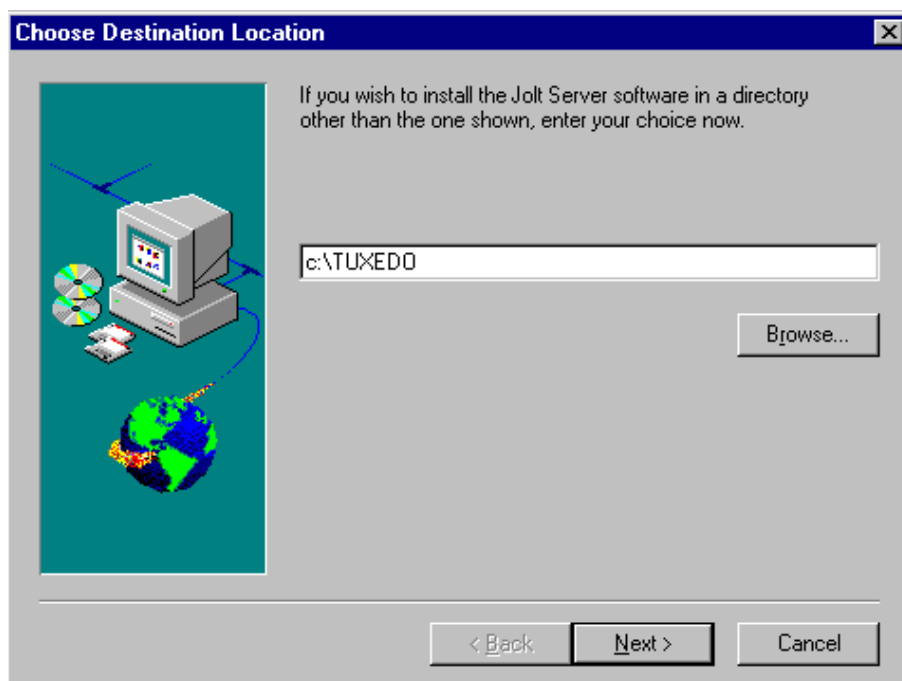
**Figure 2-7 Available Disk Space**



7. All Jolt system files are installed in directories relative to the destination directory. Jolt installs all system files in the default directory displayed on the window. If the default directory is not the directory that contains your TUXEDO system, change the directory.

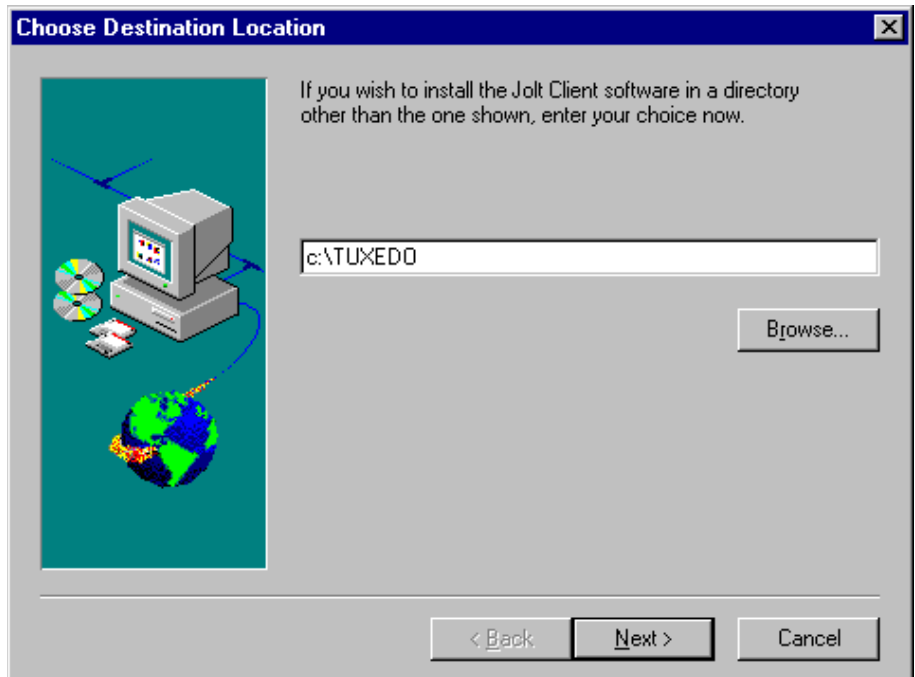
8. In Figure 2-6, Jolt Server, Jolt Client, Jolt Documentation are selected for installation. Figure 2-8 shows the window used to install the first selected module (in this case, the Jolt Server). To change the directory, select **Browse** or type the directory path. Select **Next** to continue with the installation.

**Figure 2-8 Choose Destination Location Window for Jolt Server**



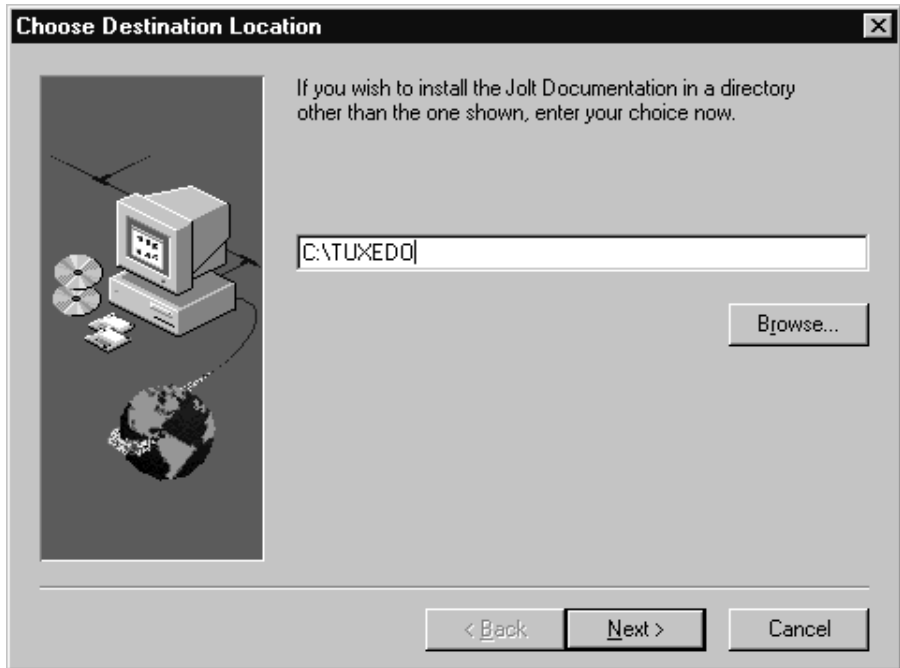
- Figure 2-9 shows the window used to install the next selected module (in this case, the Jolt Client). To change the directory, select **Browse** or type the directory path. Select **Next** to continue with the installation.

**Figure 2-9 Choose Destination Location Window for Client Software**



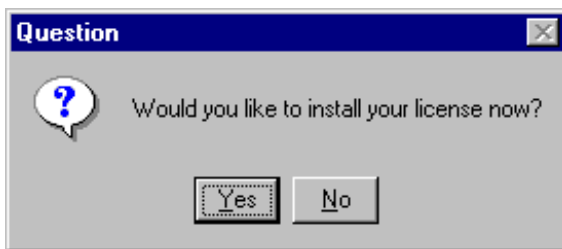
10. You are prompted to install the final module (in this case, the Jolt Documentation). To change the directory, select **Browse** or type the directory path. Select **Next** to continue with the installation.

**Figure 2-10 Choose Destination Location Window for Jolt Documentation**



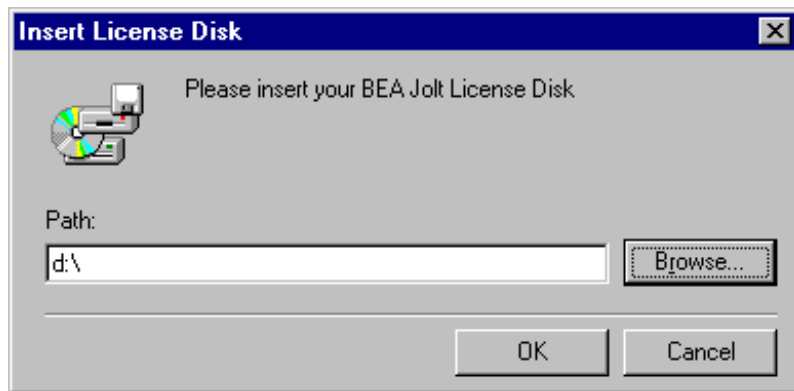
11. When the selected modules are installed, you are prompted to install your Jolt license. The prompt is shown in Figure 2-11.

**Figure 2-11 Install License Window**



12. Select **Yes** to install the Jolt license.
13. If you are installing Jolt 1.1 for TUXEDO 6.1 or 6.2, a window displays prompting you to enter your serial number and license key. If you are installing Jolt 1.1 for TUXEDO 6.3, the Insert License Disk window shown in Figure 2-12 displays, and you are prompted to insert your license disk.

**Figure 2-12 Insert License Disk Window**



14. Type the designated path or select **Browse** and select **OK**.
15. When all files are installed, the Setup Complete window displays. Select a Jolt application by choosing an icon from the window.

## Licensing your Jolt Software

The licensing scheme used by Jolt 1.1 depends on the version of TUXEDO used with Jolt. Know the following information before running this command:

- ◆ Your version of TUXEDO
- ◆ Your TUXEDO directory (TUXDIR) from the installation. This is the directory that contains the TUXEDO directories (bin, udataobj, etc.).
- ◆ Your serial number (included with your Jolt software)
- ◆ Your license key or token (included with your Jolt software)

## Licensing Jolt 1.1 for TUXEDO 6.1 and 6.2

Jolt 1.1 for TUXEDO 6.1 and 6.2 uses a combination of a serial number and a token number (or license key) to enable a license. This information is included with the Jolt software when it is shipped to you. The installation programs (`install.sh` for UNIX or `setup.exe` for NT) prompt you for the serial number and token at installation time. Refer to “UNIX System Installation Script” and “Windows NT Installation Instructions” for additional information.

Enter the serial number and token number exactly as they are displayed on the license provided by BEA.

## Licensing Jolt 1.1 for TUXEDO 6.3

Jolt 1.1 for TUXEDO 6.3 uses a digitally signed license file to enable a license. This file is provided on a floppy disk that is shipped with your Jolt software. The UNIX installation program (`install.sh`) does not install the license automatically. The NT installation program (`setup.exe`) prompts you for the location of the Jolt license file. If you provide the necessary information, the installation program installs the license file for you. If you do not install the license file during installation, follow the steps to install Jolt manually.

### UNIX LICENSING INSTRUCTIONS

1. Identify your current TUXEDO license file. This is located in `TUXDIR/udataobj/lic.txt`.
2. Make a copy of this file:  

```
cd $TUXDIR/udataobj  
cp lic.txt lic.txt.bak
```
3. Check that you have completed step 2. Verify the copy using OS-specific commands (e.g., `diff` on UNIX systems).
4. Append the contents of the Jolt license file to the TUXEDO license file:  

```
cat /dev/diskette/joltlic.txt >> lic.txt
```

### NT LICENSING INSTRUCTIONS

1. Identify your current TUXEDO license file. This is located in  
`%TUXDIR%\udataobj\lic.txt.`
2. Make a copy of this file:  
  
`cd %TUXDIR%\udataobj`  
  
`copy lic.txt lic.txt.bak`
3. Check that you have completed step 2. Verify the copy using OS-specific commands.
4. Append the contents of the Jolt license file to the TUXEDO license file:  
  
`copy lic.txt + a:\joltlic.txt`

A text editor can be used to copy and paste the contents of the Jolt license file into the TUXEDO license file.

**Note:** The digital signature is 64 characters long. Every character must match exactly or the license is not valid.

If you do not complete the steps for licensing the Jolt software during installation, you can license the software at any time by following the steps in the “Licensing Upgrades Instructions.”

## LICENSING UPGRADES INSTRUCTIONS

If you are using Jolt 1.1 on TUXEDO 6.1 or 6.2, and you need to upgrade your license, follow these step.

**Note:** These instructions are for NT. Use the UNIX equivalents for UNIX platforms.

1. Bring up an NT command prompt window.
2. Type the following:

```
SET TUXDIR=<the name of your TUXEDO directory>
SET SERNUM=<your serial number>
SET LICENSE=<your license key>
CD %TUXDIR%
CD BIN
JOUTIL -r %TUXDIR% -b -l %LICENSE% -s %SERNUM% -T SDK
```

For example, if TUXEDO is installed in the directory, /opt/BEAItuxedo, your serial number is 0123456789, and your license key is 0A1B2C3D4E5F9999, execute the following commands:

```
SET TUXDIR=/opt/BEAItuxedo
SET SERNUM=0123456789
SET LICENSE=0A1B2C3D4E5F9999
CD %TUXDIR%
CD BIN
JOUTIL -r %TUXDIR% -b -l %LICENSE% -s %SERNUM% -T SDK
```

**Note:** The “-l” above is a lowercase “L” not the numeral “1” (one).

Your Jolt executables are now properly licensed. JOUTIL responds with an error if your serial number or license key is incorrect. If you receive an error, ensure that you have the correct serial number and license key.

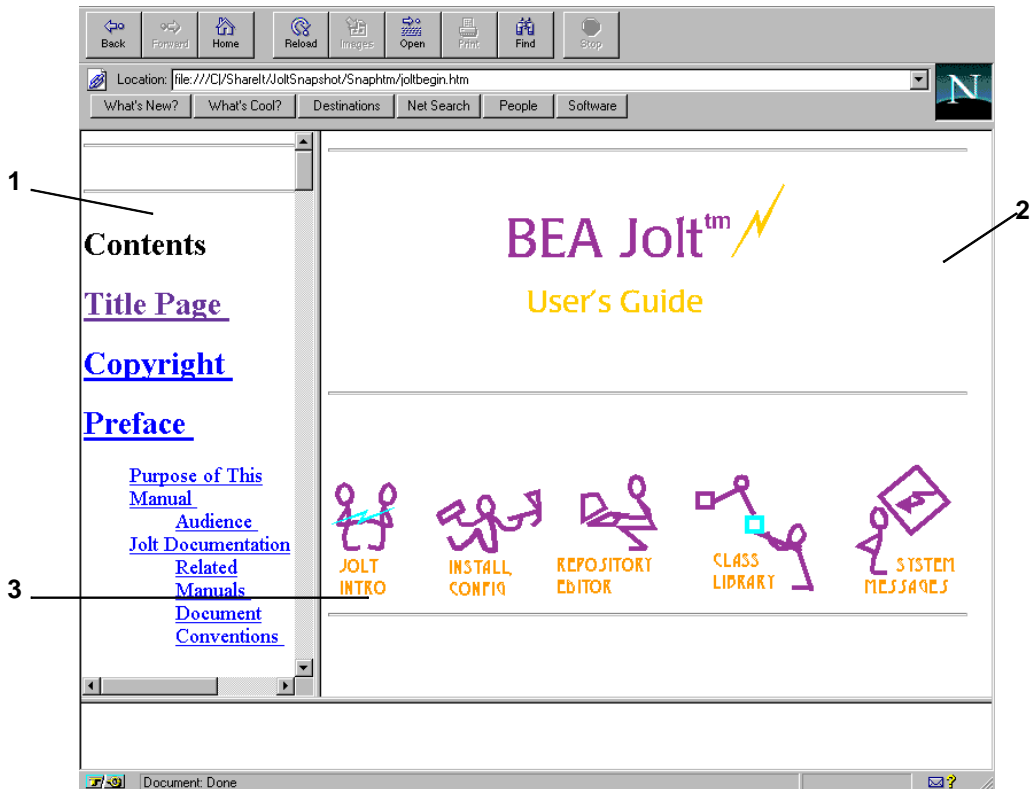


# Using the Jolt Online Documentation

Accompanying your Jolt software is an online, HTML-based, documentation set to assist you with using BEA Jolt 1.1. The Jolt product CD-ROM contains the HTML version of the *Jolt User's Guide*.

Figure 2-13 is an example of the Jolt online documentation window. Table 2-2 describes the online documentation browser application components shown in Figure 2-13.

**Figure 2-13 Jolt User's Guide Online Documentation Window**



**Table 2-2 Jolt User's Guide Online Documentation Window Parts**

<b>Part</b>	<b>Function</b>
<b>1</b> Table of Contents	View the online documentation Table of Contents. All topics are hypertext links. Selecting the topic accesses the accompanying documentation.
<b>2</b> Documentation Window	View the online documentation contents.
<b>3</b> Topic Buttons	Access a subject area by selecting the topic button: JOLT INTRO - Introduction to the Jolt product and its features. INSTALL CONFIG - Description of the Jolt installation and configuration process. REPOSITORY EDITOR - Instructions for using the Jolt Repository Editor and Bulk Loader. CLASS LIBRARY - Instructions for using the Jolt Class Library. SYSTEM MESSAGES - Jolt system message reference.

## Getting Started with the Documentation

To open the Jolt online documentation, your browser must support:

- ◆ HTML frame tags
- ◆ HTML table tags

The recommended browsers for use with the Jolt 1.1 HTML documentation release are:

- ◆ Netscape Navigator 2.02
- ◆ Microsoft Internet Explorer 3.0 or later

### Opening the Documentation Files

Follow these instructions for opening the documentation files with a specific browser.

#### NETSCAPE NAVIGATOR

Using Netscape Navigator, access the documentation using the following instructions:

1. From the menu bar of your browser, select **File**.
2. From the File menu, select **Open File** in Browser.
3. From the Open window, locate the directory where you installed your Jolt documentation files.
4. Select the `joltbegin.htm` (NT) or `joltbegin.html` (UNIX) file and select **Open**.

#### MICROSOFT INTERNET EXPLORER

Using Microsoft Internet Explorer, access the documentation using the following instructions:

1. From the menu bar of your browser, select **File**.
2. From the File menu, select **Open**.
3. Locate the directory where you installed your Jolt documentation files.
4. Open the file to the `joltbegin.htm` (NT) or `joltbegin.html` (UNIX) file.

## Printing the Documentation Files

Follow these instructions for printing the documentation files with a specific browser.

### NETSCAPE NAVIGATOR FOR WINDOWS

Using Netscape Navigator for Windows, print the documentation using the following instructions:

1. Activate the frame you want to print by using the left mouse button to select the frame.
2. From the File menu, select **Print**.

### NETSCAPE NAVIGATOR FOR UNIX

Using Netscape Navigator for UNIX, print the documentation using the following instructions:

1. Activate the frame you want to print by using the left mouse button to select the frame.
2. From the File menu, select **Print Frame**.
3. The Netscape Print window displays. Select **Print**.

### MICROSOFT INTERNET EXPLORER

Using Microsoft Internet Explorer, print the documentation using the following instructions:

1. Activate the frame you want to print by using the left mouse button to select the frame.
2. From the menu bar, select **Print**.





# 3 Configuring the Jolt System

This chapter explains how to configure Jolt 1.1 and describes the Jolt Internet Relay, Event Subscription, and security features. Readers of this chapter are assumed to be system administrators and/or application developers who have experience with the operating systems and workstation platforms on which they are going to configure TUXEDO and Jolt.

This chapter includes the following sections:

- ◆ Using the Jolt Server
- ◆ Using the Jolt Repository
- ◆ Event Subscription
- ◆ Jolt Internet Relay
- ◆ Using Sample Applications in Jolt Online Resources

# Using the Jolt Server

The Jolt Server consists of listeners and handlers.

**Jolt Server Listener (JSL).** The JSL is configured to support clients on an IP/port combination. The JSL works with the Jolt Server Handler (JSH) to provide client connectivity to the backend of the Jolt system. The JSL is administered by the same tools used to manage any resource within a BEA TUXEDO environment.

**Jolt Server Handler (JSH).** The JSH is a program that runs on a 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 Jolt system.

The system administrator's responsibilities for the server components of Jolt include:

- ◆ Determining server network addresses.
- ◆ Determining the number of Jolt clients to be serviced by one JSH (for example, if there are 10 clients per JSH and 10 JSHs, 100 clients can be connected).
- ◆ Determining the minimum and maximum number of JSHs.

## Jolt Internet Relay

The Jolt Internet Relay is a component that routes messages from a Jolt client to a JSL or JSH. This alleviates the need for the JSL/JSH and TUXEDO to run on the same machine as the Web server. The Jolt Relay (JRLY) is not required to be a TUXEDO server or a TUXEDO client. It is a stand-alone piece of software that routes the Jolt messages to the JSL or JSH. Refer to the “Jolt Internet Relay” in this chapter.

## Security and Encryption

Authentication and key exchange data that are transmitted between Jolt clients and the JSL/JSH are encrypted using DES encryption. All subsequent exchanges are encrypted using RC4 encryption. International packages use 40-bit key; domestic packages use 128-bit key.

Programs using the 128-bit encryption cannot be exported outside of the United States; therefore, clients cannot be outside of the United States. Customers with intranets extending beyond the United States cannot use this mode of encryption if any internal clients are outside of the United States.

## Starting the Jolt Server

Type `tmloadcf` and `tmboot -y` to start all administrative and server processes. (The prompt only displays when the command is entered with none of the limiting options). See the *TUXEDO System Administration Guide* for information on `tmloadcf` and `tmboot`.

**Note:** TUXEDO monitors the JSL and restarts it in the event of a failure. When TUXEDO restarts the listener process, the following occurs:

- ◆ 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).



# Configuring the Jolt Server

The Jolt Server Listener (JSL) is a TUXEDO server responsible for distributing connection requests from Jolt to the JSH. TUXEDO must be running on the host machine where the JSL and JREPSVR is located.

To configure the JSL, you must modify the UBBCONFIG file. For information regarding TUXEDO configuration, refer to the *TUXEDO Administration Guide*. Listing 3-1 shows relevant portions of the UBBCONFIG file.

**Listing 3-1 UBBCONFIG File**

---

```
*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"
```

---

Change the following sections of the UBBCONFIG file.

**Table 3-1 UBBCONFIG File Sections**

Section	Parameters to be specified
*MACHINE	MAXWSCLIENTS
*GROUPS	LMID, GRPNO
*SERVERS	SRVGRP, SRVID, CLOPT

The parameters shown in Table 3-1 are the only parameters that should be designated for the Jolt Server groups and Jolt Servers. You do not need to specify any other parameters.

**Note:** Ensure that Resource Managers are not assigned as a default value for all groups in the \*GROUPS section of your UBBCONFIG file. This will assign a Resource Manager to the JSL and you will receive an error during `tmboot`. In the \*SERVERS section, default values for RESTART, MAXGEN, etc., are acceptable defaults for the JSL.

## \*MACHINES Section

The `MAXWSCLIENTS` parameter is required in the \*MACHINES section for the configuration file and applies to specific machines. The Jolt Server and /WS use `MAXWSCLIENTS` in the same way. `MAXWSCLIENTS` communicates the number of accesser slots to reserve for Jolt and /WS clients to TUXEDO. For example, if 200 slots are configured for `MAXWSCLIENTS`, this number configures TUXEDO for the total number of remote clients used by Jolt and /WS.

Specify `MAXWSCLIENTS` in the configuration file. If it is not specified, the default is 0.

## \*GROUPS Section

A \*GROUPS entry is required for the group that includes the Jolt Server Listener (JSL). The group name is selected by the application.

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 30,000 in the \*GROUPS section.

## \*SERVERS Section

Clients connect to Jolt applications through the 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 the multiple clients 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 port number is 8000 and there are a maximum of three JSH processes, the JSH processes use ports 8001, 8002, and 8003.

**Note:** Port numbers used by the JSHs are sequentially incremented by one numeric digit after the JSL port number. If JSL is using port number 8000, its JSHs use 8001, and so on. Misconfiguration of the subsequent JSL results in a port number collision.

Each handler uses a multiplexing scheme on its designated port to support multiple clients concurrently on one port.

TUXEDO parameters including RESTART, RQADDR, and REPLYQ can be used with the JSL. See the *TUXEDO Administration Guide* for additional information regarding run-time parameters. Enter the following parameters:

1. To identify the SRVGRP parameter, type the previously defined group name value from the \*GROUPS section.
2. To indicate the SRVID, type a number between 1 and 30,000 that identifies the server within its group.
3. Verify that the syntax for the CLOPT parameter is as follows:

```
CLOPT= "-- -n 0x0002PPPPNNNNNNNN -d /dev/tcp -m2 -M4 -x10"
```

**Note:** The CLOPT parameters may vary. Refer to Table 3-2 for pertinent command-line information.

4. If necessary, type the optional parameters:
  - ◆ Type the SEQUENCE parameter to determine the order that the servers are booted.
  - ◆ Specify Y to permit release of the RESTART parameter.
  - ◆ Type 0 to permit an infinite number of server restarts using the GRACE parameter.

**Table 3-2 Command Line Options**

Command Line Option	Description
<code>[-c <i>connection_mode</i>]</code>	<p>Allowed connection modes from clients:</p> <p>RETAINED - the network connection is retained for the full duration of a session.</p> <p>RECONNECT - the client establishes and brings down a connection when an idle timeout is reached, reconnecting for multiple requests within a session.</p> <p>ANY - the server allows a client to request either a RETAINED or RECONNECT type of connection for a session. Default is ANY. (Optional)</p>
<code>[-d <i>device_name</i>]</code>	<p>The device for platforms using the Transport Layer Interface. There is no default. (Required; optional for sockets)</p>
<code>[-H <i>external_netaddr</i>]</code>	<p>The external netaddr is the network address Jolt 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 0x0002MMMMddddddd and JSH network address is 0x00021111ffffffffff, 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. (Optional for JSL in TUXEDO 6.3)</p>
<code>[-I <i>init-timeout</i>]</code>	<p>The time (in seconds) that a Jolt client is allowed to complete initialization through the JSH before it is timed out by the JSL. Default is 60 seconds. (Optional)</p>
<code>[-m <i>minh</i>]</code>	<p>The minimum number of JSHs that are available in conjunction with the JSL at one time. The range of this parameter is between 0 and 255. Default is 0. (Optional)</p>
<code>[-M <i>maxh</i>]</code>	<p>The maximum number of JSHs that are available in conjunction with the JSL at one time. The range of this parameter is between 1 and 4096. If this option is not specified, the parameter defaults to MAXWSCLIENTS divided by the rounded-up -x multiplexing factor. (Optional)</p>

**Table 3-2 Command Line Options**

Command Line Option	Description
<code>-n netaddr</code> (TUXEDO 6.1 and 6.2)	<p>Network address used by the Jolt Listener for Jolt 1.1 with TUXEDO 6.1 and 6.2.</p> <p>Indicate the network address where the clients connect to the JSL. This is a required parameter and is the contact point used by Java workstation clients to access the application.</p> <p>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.</p> <p>The network address is composed of 1) an initial two digits indicating hexadecimal characters, followed by three groups of numbers indicating 2) protocol, 3) port number, and 4) IP address.</p> <p>For example: 0x            0002    PPPP   NNNNNNNN</p> <p>There is no default. (Required)</p>
<code>-n netaddr</code> (for TUXEDO 6.3)	<p>Network address used by the Jolt listener for Jolt 1.1 with TUXEDO 6.3.</p> <p>TCP/IP addresses may be specified in the following forms:</p> <p><code>//host.name:port_number</code></p> <p><code>//#. #. #. #:port_number</code></p> <p>In the first format, the domain finds an address for hostname 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.</p> <p>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.</p>

**Table 3-2 Command Line Options**

Command Line Option	Description
	<p>In both of the above formats, <code>port_number</code> is the TCP port number at which the domain process will listen for incoming requests. <code>port_number</code> can either be a number between 0 and 65535 or a name.</p> <p>If <code>port_number</code> 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 between 0 and 9 or a letter between A and F (case insensitive). The hexadecimal format is useful for arbitrary binary network addresses such as IPX/SPX or TCP/IP.</p> <p>There is no default. (Required)</p>
<code>[-T <i>Client-timeout</i>]</code>	<p>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.</p> <p>When the <code>-c ANY</code> or <code>-c RECONNECT</code> option is used, always specify <code>-T</code> with an idle timeout value. If <code>-T</code> 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.</p> <p>If a parameter is not specified, the default is no timeout. (Optional)</p>
<code>[-w <i>JSH</i>]</code>	<p>The Jolt Server Handler is indicated by this command line option. Default is JSH. (Optional)</p>
<code>[-x <i>mpx-factor</i>]</code>	<p>A parameter used to control the degree of multiplexing within each JSH process. This is the number of clients that one JSH can service. Default value is 10. (Optional)</p>
<code>[-z 40 128]</code>	<p>When establishing a network link between a Jolt client and the JSH, allow encryption up to this level. 40 and 128 specify the length (in bits) of the encryption key. The default value is 0.</p>

## Shutting Down the Jolt Server

All shutdown requests to the Jolt servers are initiated by the TUXEDO command, `tmsshutdown -y`. During shutdown:

- ◆ No new client connections are accepted.
- ◆ All current client connections are terminated. TUXEDO will rollback in-flight transactions. Each client receives an error message indicating that the service is unavailable.

## Using the Jolt Repository

The Jolt Repository contains TUXEDO service definitions that allow the Jolt clients to access TUXEDO services. The Jolt Repository files included with the installation contain services definitions used internally by Jolt. See Chapter 5, “Using the Jolt Repository Editor,” for detailed instructions on how to add definitions to the application services.

# Configuring the Jolt Repository

To configure the Jolt Repository, modify the application UBBCONFIG file. The UBBCONFIG file is an ASCII version of the TUXEDO configuration file. Create a new UBBCONFIG file for each application. See the *TUXEDO Reference Manual* for information regarding the syntax of the entries for the file. Listing 3-2 shows relevant portions of the UBBCONFIG file.

**Listing 3-2 Sample of UBBCONFIG File**

```
*GROUPS
JREPGRP          GRPNO=94  LMID=SITel
*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"
```

**Note:** For UNIX systems, use the slash (/) when setting the path to the jrepository file. For NT systems, use the backslash (\) and specify the drive name (e.g., c:\app\repository).

Change the following sections of the UBBCONFIG file:

**Table 3-3 UBBCONFIG File**

Section	Parameters to be specified
*GROUPS	LMID, GRPNO
*SERVERS	SRVGRP, SRVID



### \*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.

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 30,000 in the \*GROUPS section.

### \*SERVERS Section

The Jolt Repository server, `JREPSVR`, contains services for access and editing the Repository. Multiple `JREPSVR` instances share repository information through a shared file. Include `JREPSVR` in the \*SERVERS section of the `UBBCONFIG` file.

1. Indicate a new server identification (e.g., 98) with the `SRVID` parameter.
2. Specify the `-w` flag for one `JREPSVR` to ensure that you can edit the Repository. The Repository is read-only without this flag.

**Note:** You must install only one writable `JREPSVR` (i.e., only one `JREPSVR` with the `-w` flag). Multiple read-only `JREPSVRs` may be installed on the same host.

3. Type the `-P` flag to specify the path of the repository file. An error message displays in the `TUXEDO ULOG` file if the argument for the `-P` flag is not entered.
4. Add the file pathname of the Repository file (e.g., `/app/jrepository`).
5. Boot the `TUXEDO` system using the `tmloadcf` command (e.g., `tmloadcf -y ubbconfig`) and `tmboot` command. See the *TUXEDO Administration Guide* for information on `tmloadcf` and `tmboot`.

## Repository File

A Repository file, `jrepository`, is available with Jolt 1.1. This file includes `bankapp` services and the Repository services that you can modify, test, and delete using the Repository Editor.

Start with the `jrepository` file provided with the installation, even if you are not going to test the `bankapp` application with Jolt. Delete the `bankapp` packages or services that are not needed.

The pathname of the file must match the argument of the `-P` option.



**Warning:** Do not modify the Repository files manually or you will not be able to use the Repository Editor. Although the `jrepository` file can be modified and read with any text editor, the Jolt system does not have integrity checks to ensure that the file is in the proper format. Any manual changes to the `jrepository` file may not be detected until runtime. See “Using the Jolt Repository Editor” for additional information.

## Initializing Services Using TUXEDO and the Repository Editor

You must initially define the TUXEDO services using TUXEDO and Jolt in order to make the Jolt services available to the client.

1. Build the TUXEDO server containing the service. See the *TUXEDO Administration Guide* or *TUXEDO Programmer's Guide* for additional information on the following:
  - ◆ Building the TUXEDO applications/server
  - ◆ Editing the `UBBCONFIG` file
  - ◆ Updating the `TUXCONFIG` file
  - ◆ Administering the `tmboot` command
2. Access the Jolt Repository Editor. See Chapter 5, “Using the Jolt Repository Editor,” in this book for additional information on the following:
  - ◆ “Adding a Service”
  - ◆ “Saving Your Work”
  - ◆ “Testing a Service”
  - ◆ “Exporting/Unexporting Services”

# Event Subscription

Jolt Event Subscription is used to receive event notifications from either TUXEDO services or other TUXEDO clients:

**Unsolicited Event Notifications.** These are notifications that a Jolt client receives as a result of a TUXEDO client or service subscribing to unsolicited events, and a TUXEDO client issuing broadcast (using either a `tpbroadcast()` or a directly targeted message via a `tpnotify()` ATMI call).

**Brokered Event Notifications.** These notifications are received by a Jolt client via the TUXEDO Event Broker. The notifications are only received when both Jolt clients subscribes to an event and any TUXEDO client or server issues system posted event or a `tpost()` call.

## Configuration

Configure the TUXEDO TMUSREVT server and modify the application UBBCONFIG file. Listing 3-3 shows the relevant portions TMUSREVT parameters in the UBBCONFIG file. See the *TUXEDO Programmer's Guide* for information regarding the syntax of the entries for the file.

### Listing 3-3 UBBCONFIG File

---

```
TMUSREVT      SRVGRP=EVBGRP1  SRVID=40          GRACE=3600
               ENVFILE="/usr/tuxedo/bankapp/TMUSREVT.ENV"
               CLOPT="-e tmusrevt.out -o tmusrevt.out -A --
               -f /usr/tuxedo/bankapp/tmusrevt.dat"
               SEQUENCE=11
```

---

Change the following sections of the UBBCONFIG file:

**Table 3-4 UBBCONFIG File**

Section	Parameters to be specified
*SERVERS	SRVGRP, SRVID

## Filtering TUXEDO FML or VIEW Buffers

“Filtering” is a process that allows you to customize a subscription. If you require additional information about the TUXEDO Event Broker, subscribing to events, or filtering, refer to the *BEA TUXEDO Programmer’s Guide, Volume 1*.

In order to filter TUXEDO FML or VIEW buffers, the field definition file must be available to TUXEDO at runtime.

**Note:** There are no special requirements for filtering STRING buffers.

### FML BUFFER EXAMPLE

Listing 3-4 shows an example using the FML buffer. The FML field definition table is made available to TUXEDO by setting the FIELDTBLS and FLDTBLDIR variables.

To filter a field found in the `my.flds` file:

1. Copy the `my.flds` file to `/usr/me/bankapp` directory.
2. Add `my.flds` to the FIELDTBLS variable in the `TMUSREVT.ENV` file as shown in Listing 3-4:

### Listing 3-4 FIELDTBLS Variable in the TMUSREVT.ENV File

---

```
FIELDTBLS=Usysflds,bank.flds,credit.flds,event.flds,my.flds
FLDTBLDIR=/usr/tuxedo/me/T6.2/udataobj:/usr/me/bankapp
```

---

If `ENVFILE="/usr/me/bankapp/TMUSREVT.ENV"` is included in the definition of the `UBBCONFIG` file (shown in Listing 3-3), the `FIELDTBLS` and `FLDTBLDIR` definitions are taken from the `TMUSREVT.ENV` file and not from your environment variable settings.

If you remove the `ENVFILE="/usr/me/bankapp/TMUSREVT.ENV"` definition, the `FIELDTBLS` and `FLDTBLDIR` definitions are taken from your environment variable settings. The `FIELDTBLS` and `FLDTBLDIR` definitions must be set to the appropriate value prior to booting the TUXEDO system.

For additional information on event subscriptions and the Jolt Class Library, refer to Chapter 6, “Using the Jolt Class Library.”

## Jolt Internet Relay

The Jolt Internet Relay is a component that routes messages from a Jolt client to a JSL or JSH. This eliminates the need for the JSH and TUXEDO to run on the same machine as the Web server (generally considered as insecure). The Jolt Relay (JRLY) is not a TUXEDO server or a TUXEDO client. It is a stand-alone program that routes the Jolt messages from the Internet to the JSL or JSH.

The Jolt Internet Relay consists of two components illustrated in Figure 3-1.

- ◆ **Jolt Relay (JRLY).** The JRLY is not a TUXEDO client or server. It is a stand-alone software component. It requires only minimal configuration to allow it to work with Jolt clients.
- ◆ **Jolt Relay Adapter (JRAD).** The JRAD is a TUXEDO application server, but does not include any TUXEDO services. It requires command line arguments to allow it to work with the JSL and the TUXEDO system.

**Note:** The Jolt relay is transparent to Jolt clients and Jolt servers. A Jolt server can simultaneously connect clients directly to the Jolt Server (intranet clients), or via the Jolt Relay (Internet clients).

Figure 3-1 Jolt Internet Relay Path

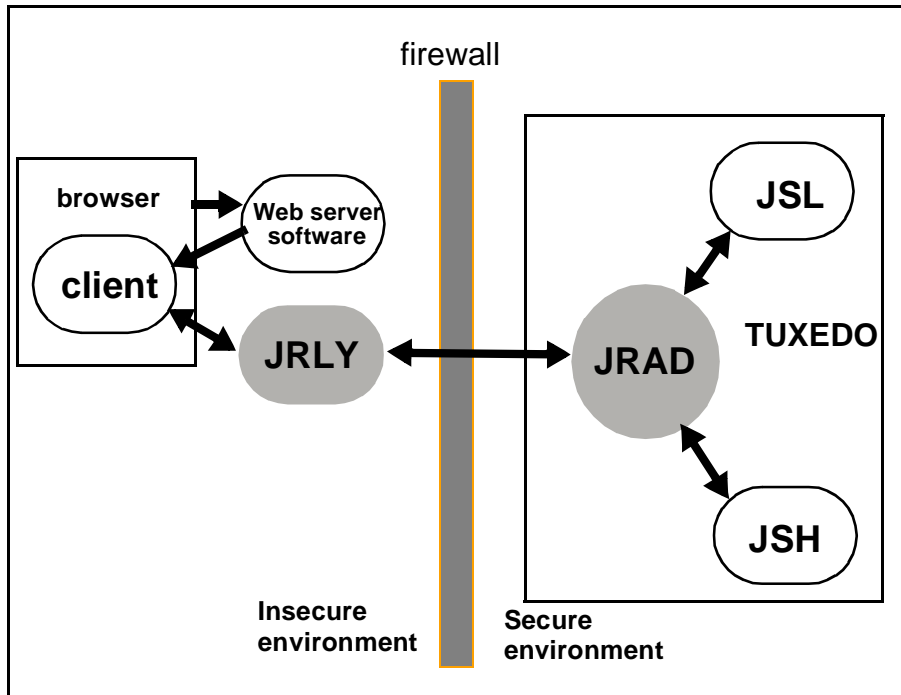


Figure 3-1 shows how a browser connects to the Web server software and downloads the Jolt applets. The Jolt applet or client connects to the JRLY on the Web server machine. The JRLY forwards the Jolt messages across the firewall to the JRAD. The JRAD selectively forwards messages to the JSL or appropriate JSH.

## Jolt Relay (JRLY)

The JRLY (front-end relay) process may 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 unable to connect to the JRAD, the client is denied access and is disconnected. A warning is written to the JRLY error log file.

### Starting the JRLY

The JRLY process is started by typing the command name at a system prompt.

```
jrly -f <config_file_path>
```

If the configuration file does not exist or cannot be opened, the JRLY prints an error message. Refer to Appendix B for the Jolt Relay messages.

If the JRLY is unable to start, it writes a message to standard error and attempts to log the startup failure in the error log if possible, then exit.

### JRLY Configuration File

The format of the configuration file is a TAG=VALUE format. Blank lines or lines starting with a “#” are ignored. Refer to Listing 3-5 for an example of the formal specifications of the configuration file.

---

**Listing 3-5    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>
```

---

Refer to Listing 3-6 for an example of the JRLY configuration file. The `CONNECT` line specifies the IP address and port number of JRAD machine.

---

**Listing 3-6 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
```

---

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

The format for host names and port numbers are shown in Table 3-5.

**Table 3-5 Host Name and Port Number Formats**

Host Name/Port Number	Descriptions
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 (JRAD)

The JRAD (back-end relay) is a TUXEDO system server. The JRAD server may or may not be located on the same TUXEDO host machine (in SHM mode) and server group that the JSL server it is connected to.

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

### Starting the JRAD

Type `tmloadcf` and `tmboot` to start the JRAD.

Configuration entry in the `UBBCONFIG` is described in the “JRAD Configuration” section.

### JRAD Configuration

A single JRAD process can only be connected to a single JRLY. A JRAD can only be configured to communicate with one JSL and its associated JSHs. However, multiple JRADs can be configured to communicate with one JSL. The `CLOPT` parameter for the TUXEDO servers must be included in the `UBBCONFIG` file. For additional information about the `CLOPT` parameters, refer to Table 3-2 and Table 3-6.

**Table 3-6 JRAD CLOPT Parameter Descriptions**

CLOPT Parameter	Description
<code>-l &lt;hexadecimal format&gt;</code>	Port to listen for the JRLY to connect on behalf of the client.
<code>-c &lt;hexadecimal format&gt;</code>	The address of the corresponding JSL to which JRAD connects.

**Note:** The format is `0x0002PPPPNNN`. Refer to the *Jolt 1.1 Release Notes* for additional information on JRAD.

Listing 3-7 shows the sample UBBCONFIG file.

**Listing 3-7 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**

There are several networked components that need to be configured to work together when configuring a Jolt Internet Relay. Prior to configuration, review the criteria required in Table 3-7 and record the information. This will help minimize the possibility of misconfiguration.

**Table 3-7 Jolt Internet Relay Network Address Configuration Criteria**

JRLY	JRAD	JSL
LISTEN: <Location where the clients connect>	-l: <Location of where the listener connects 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>	

# Using Sample Applications in Jolt Online Resources

You can access sample code that can be modified for use with BEA Jolt through the BEA Jolt product Web page at:

<http://www.beasys.com/products/jolt/index.htm>

These samples demonstrate and utilize Jolt features and functionality.

Other Web sites with Java-related information include:

- ◆ [Javasoft Home Page \(http://www.javasoft.com/\)](http://www.javasoft.com/)
- ◆ [Deja News Home Page \(http://www.dejanews.com/\)](http://www.dejanews.com/), an archive of Java-related newsgroups
- ◆ [Java Programmers FAQ \(http://www.best.com/~pvd1/javafaq.txt\)](http://www.best.com/~pvd1/javafaq.txt)
- ◆ In addition, the newsgroups in the comp.lang.java hierarchy contain lists of past articles and communications regarding Java, and is a valuable source of archival material.



# 4 Bulk Loading TUXEDO Services

This section covers the following topics:

- ◆ Introduction to the Bulk Loader
- ◆ Getting Started Using the Bulk Loader
- ◆ Syntax of the Bulk Loader Data Files
- ◆ Troubleshooting
- ◆ Sample Bulk Load Data

## Introduction to the Bulk Loader

As a systems administrator, you may have an existing TUXEDO application with multiple TUXEDO services. Manually creating these definitions to the repository database may take hours to complete.

Using the program, `jblld`, the bulk loader utility reads the specified text file consisting of the TUXEDO service definitions and bulk loads them into the Jolt repository. The services are loaded to the repository database in one “bulk load.” After the services have populated the Jolt Repository, you may edit services, create new services, and group services using the Jolt Repository Editor.

See Chapter 5, “Using the Jolt Repository Editor,” for information about using the Jolt Repository Editor.

# Getting Started Using the Bulk Loader

Since `jbld` 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 (e.g., `/opt/BEA/jolt/classes`) is located. If it is not set, the Java Virtual Machine cannot locate any Jolt classes.

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` will automatically prompt the user for passwords.

The bulk loader utility gets its input from command-line arguments and from the input file.

## Using UNIX

To activate the bulk loader using UNIX:

1. Type the following at the prompt:

```
setenv CLASSPATH <pathname>

java bea.joltadm.jbld [-n] [-u usrname] [-r usrrole]
<>//host:port inputfile>
```

2. Type your user password and application password, if required.

## Using Windows NT

1. To activate the bulk loader using Windows NT, type:

```
C:\> set CLASSPATH=C:<pathname>

C:\> java bea.jolt.jbld [-n][ -u usrname] [-r usrrole]
<>//host:port> <filename>
```

2. Type your user password and application password (if required) and press **Enter**.

## Command Line Options

Table 4-1 describes the bulk loader command-line options.

**Table 4-1 Command Line Options**

Option	Description
-u <code>username</code>	Specifies the user name (default is your account name). (Mandatory if required by security)
-r <code>usrrole</code>	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)
/ / <code>host:port</code>	Specifies the JRLY or JSL address (host name and IP port number). (Mandatory)
<code>filename</code>	Specifies the file containing the service definitions. (Mandatory)

## About the Bulk Load File

The bulk load file is a text file that defines services and their associated parameters. The bulk loader loads the services defined in the bulk loader file into the repository using the package name, “BULKPKG.” If a bulk load has been performed, the “BULKPKG” package exists in the repository. If another load is performed from a bulk loader file, all the services in the original “BULKPKG” are deleted. A new “BULKPKG” package is created with the services from the new bulk loader file.

If a service exists in a package other than “BULKPKG,” the bulk loader reports the conflict and does not load a service from the bulk loader file into the repository. Use the Repository Editor to remove duplicate services and load the bulk loader file again. See Chapter 5, “Using the Jolt Repository Editor,” for additional information.

# Syntax of the Bulk Loader Data Files

Each service definition consists of services 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` reads the service definitions from a text file. While using the keywords, follow the guidelines in Table 4-2.

Table 4-2 Guidelines for Using Keywords

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

**Table 4-2 Guidelines for Using Keywords**

Guideline	Example
The input file may contain multiple service definitions.	<pre> service=INQUIRY &lt;service keywords and values&gt; service=DEPOSIT &lt;service keywords and values&gt; service=WITHDRAWAL &lt;service keywords and values&gt; service=TRANSFER &lt;service keywords and values&gt; </pre>
Each service definition consists of multiple keywords and values.	<pre> service=deposit export=true inbuf=VIEW32 outbuf=VIEW32 inview=INVIEW outview=OUTVIEW </pre>

## 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 of the 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. Refer to Table 4-3 for a list of the service keywords and values.

Next, specify all the parameters associated with the service. Following each of the `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** Correct Example of 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=” keyword. For more information about services, see Chapter 5, “Using the Jolt Repository Editor.”

**Note:** Services using CARRAY or STRING buffer types should only have one parameter in the service. The recommended parameter name for a CARRAY service is “CARRAY” with “carray” as the data type. For a STRING service, the recommended parameter name is “STRING” with “string” as the data type. See Chapter 5, “Using the Jolt Repository Editor,” for more information.

To review the service-level keywords and values, see Table 4-3.

**Table 4-3 Service Keywords and Values**

<b>Keyword</b>	<b>Value</b>
service	Any TUXEDO service name
export	true or false (default is false)
inbuf/outbuf	Select one of these buffer types: FML FML32 VIEW VIEW32 STRING CARRAY
inview	Any view name for input parameters (optional; only if VIEW or VIEW32 buffer type is used)
outview	Any view name for output parameters (optional)

## Using Parameter-Level Keywords and Values

A parameter begins with the “param=” keyword followed by a number of parameter keywords until another “param” or “service” keyword, or end-of-file is encountered. The parameters can be in any order after the “param” keyword.

See Chapter 5, “Using the Jolt Repository Editor,” for more information about parameters.

To review the parameter-level keywords and values, see Table 4-4.

**Table 4-4 Parameter 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 Repository Editor to format test screens.

# Troubleshooting

If you encounter any problems using the bulk loader utility, see Table 4-5. For a complete list of bulk loader utility error messages and solutions, see Appendix B, “System Messages.”

**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
the data type is invalid	check to ensure that the parameter is using a valid data type

# Sample Bulk Load Data

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

---

**Listing 4-2 Sample Bulk Load Data**

---

```
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
```

---





# 5 Using the Jolt Repository Editor

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

This chapter gives detailed information on the following areas:

- ◆ Introduction to the Repository Editor
- ◆ Getting Started
- ◆ Main Components of the Repository Editor
- ◆ Setting Up Packages and Services
- ◆ Grouping Services Using the Package Organizer
- ◆ Modifying Packages/Services/Parameters
- ◆ Making a Service Available to the Jolt Client
- ◆ Testing a Service
- ◆ Troubleshooting



# Introduction to the Repository Editor

The Repository is used internally by Jolt to translate Java parameters to a TUXEDO type buffer. The Repository Editor is available as a downloadable Java applet. When a 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 Jolt client.

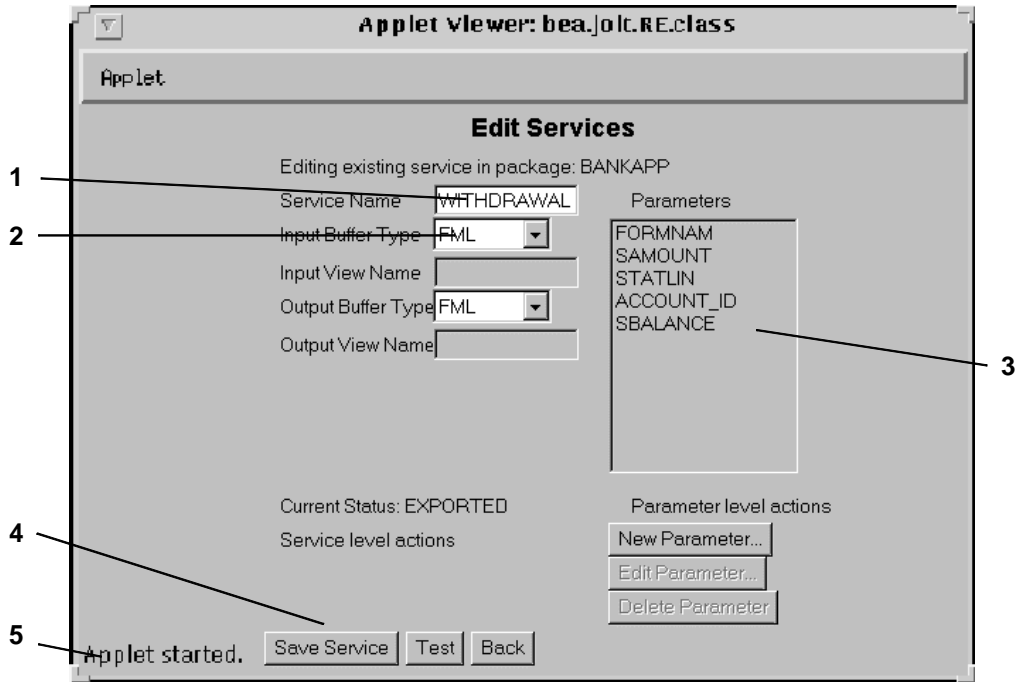
The following list describes each section in this chapter:

- ◆ “Getting Started” for information about starting the Repository Editor, logging on, and exiting the system.
- ◆ “Main Components of the Repository Editor” for information about what comprises a package, service, and parameter.
- ◆ “Setting Up Packages and Services” for information about creating packages, services, and parameters.
- ◆ “Grouping Services Using the Package Organizer” for information about moving services between packages and organizing the services.
- ◆ “Modifying Packages/Services/Parameters” for information about editing and deleting packages, services, and parameters.
- ◆ “Making a Service Available to the Jolt Client” for information about exporting and unexporting services and making the services available for use.
- ◆ “Testing a Service” for information about testing a service and its parameters.
- ◆ “Troubleshooting” for information about potential problems and solutions when using the Repository Editor.

## Repository Editor Window

Repository Editor windows contain entry fields, scrollable displays, command buttons, status, and radio buttons. Figure 5-1 illustrates the parts of a sample window.

**Figure 5-1 Sample Repository Editor Window**



## Repository Editor Window Description

Table 5-1 details the parts of the Repository Editor window example in Figure 5-1.

**Table 5-1 Repository Editor Window Parts**

Part	Function
<b>1</b> entry fields	Enter text, numbers, or alphanumeric characters such as service names, server names, or port numbers.
<b>2</b> scrollable display	View lists that extend beyond the display using a button.
<b>3</b> 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.
<b>4</b> command buttons	Activate an operation such as display the Packages window, Services window, or Package Organizer.
<b>5</b> status	View the current status of the Repository Editor service or package.
<b>6</b> radio buttons ( <i>not illustrated in Figure 5-1</i> )	Select one of a number of options. Only one of the buttons can be activated at a time.

# Getting Started

Before starting the Repository Editor, make sure that you have installed all necessary Jolt software. To use the Repository Editor, you must:

- ◆ Start the Repository Editor
- ◆ Log on to the Repository Editor

**Note:** For information on exiting the Repository Editor when you are finished inputting information, refer to “Exiting the Repository Editor” in this chapter.

Start the Repository Editor from either the JavaSoft `appletviewer` or from your Web browser.

## Starting the Repository Editor Using `appletviewer`

To start the editor using the JavaSoft `appletviewer`:

1. Set the `CLASSPATH` to include the Jolt class directory.
2. If loading the applet from a local disk, type the following at the URL location:

```
appletviewer <full-pathname>/RE.html
```

If loading the applet from the Web server, type the following at the URL location:

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

3. Press **Enter**. The Repository Editor login window displays.

## Starting the Repository Editor Using Your Web Browser

To start the Repository Editor from a local file:

1. Set the `CLASSPATH` to include the Jolt class directory.
2. Type the following:

```
file:<full-pathname>/RE.html
```

To start the Repository Editor from a Web server:

1. Ensure that the `CLASSPATH` does not include the Jolt class directory
2. Unset the `CLASSPATH`.
3. Type the following:

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

**Note:** Modify the `applet codebase` parameter in `RE.html` to match your Jolt Java classes directory.

4. Press **Enter**. The Repository Editor login window displays.

## Logging on to the Repository Editor

After starting the Jolt Repository Editor, follow the directions to log on:

1. Type the name of the server machine designated as the “access point” to the TUXEDO application and select the port number text field.
2. Type the port number and press **Enter**. The system validates the server and port information.

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

3. Type the TUXEDO Application Password and press **Enter**. Based on the authentication level, type the remaining information.

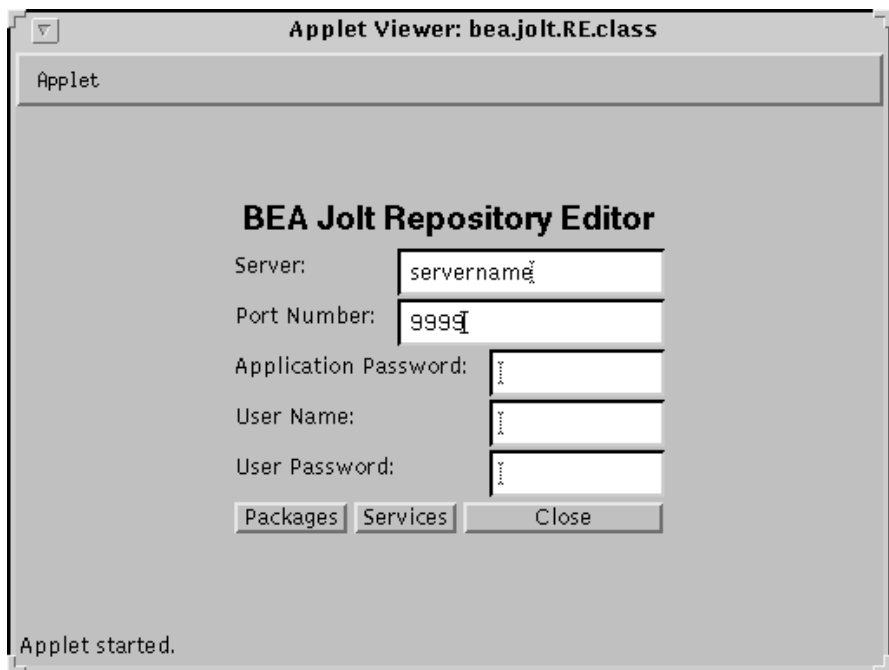
4. Type the TUXEDO user name and press **Tab**.
5. Type the TUXEDO user password and press **Enter**.

**Note:** The Jolt 1.1 Repository Editor uses the hardcoded `joltadmin` for the user role.

The **Packages** and **Services** options are activated.

Figure 5-2 is an example of the Repository Editor logon window.

**Figure 5-2 Repository Editor Logon Window**



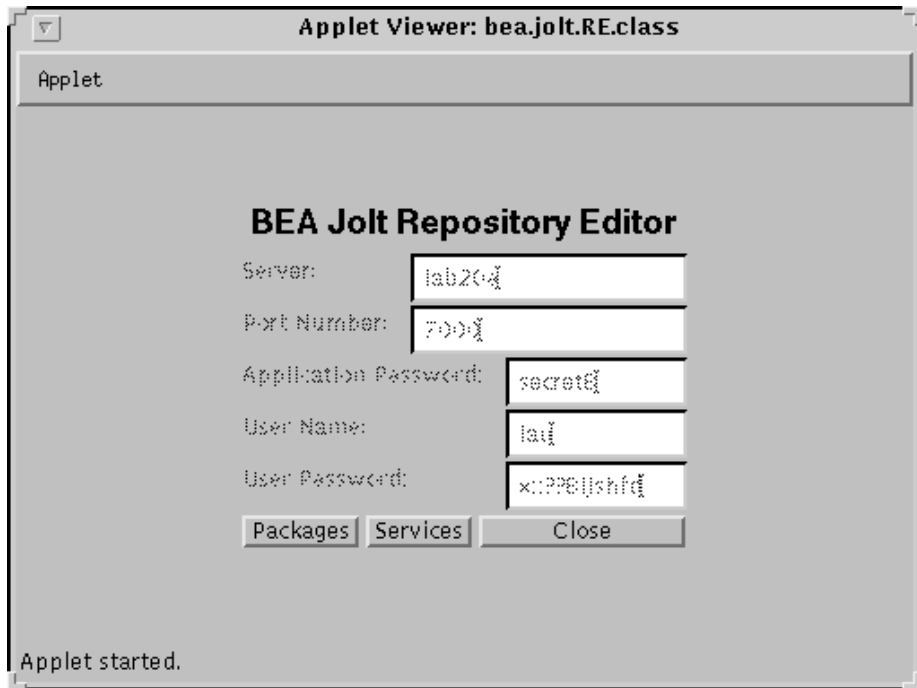
## Repository Editor Logon Window Description

The following listing details the Repository Editor logon window.

Option	Description
Server	Type the server name.
Port Number	Type the port number in decimal value.
	<b>Note:</b> 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 TUXEDO application.
Application Password	TUXEDO administrative password text entry.
User Name	TUXEDO user identification text entry. The first character must be an alpha character.
User Password	TUXEDO password text entry.
Packages	Accesses the Packages window. (Enabled after the logon.)
Services	Accesses the Services window. (Enabled after the logon.)

## Exiting the Repository Editor

Exit the Repository Editor when you are finished adding, editing, testing, or deleting packages, services, and parameters. Figure 5-3 is an example of the Repository Editor window before exiting. Only **Packages**, **Services**, and **Close** are enabled. All text entry fields are disabled.

**Figure 5-3 Example of the Repository Editor Logon Window Before Exiting**

To exit the Repository Editor:

1. Select **Back** from a previous window to return to the Logon window.
2. Select **Close** to terminate the connection with the server. The Repository Editor Logon window continues to display with disabled fields.
3. Select **Close** from your browser menu to remove the window from your screen.



# Main Components of the Repository Editor

The Repository Editor allows you to add, modify, or delete any of the following components:

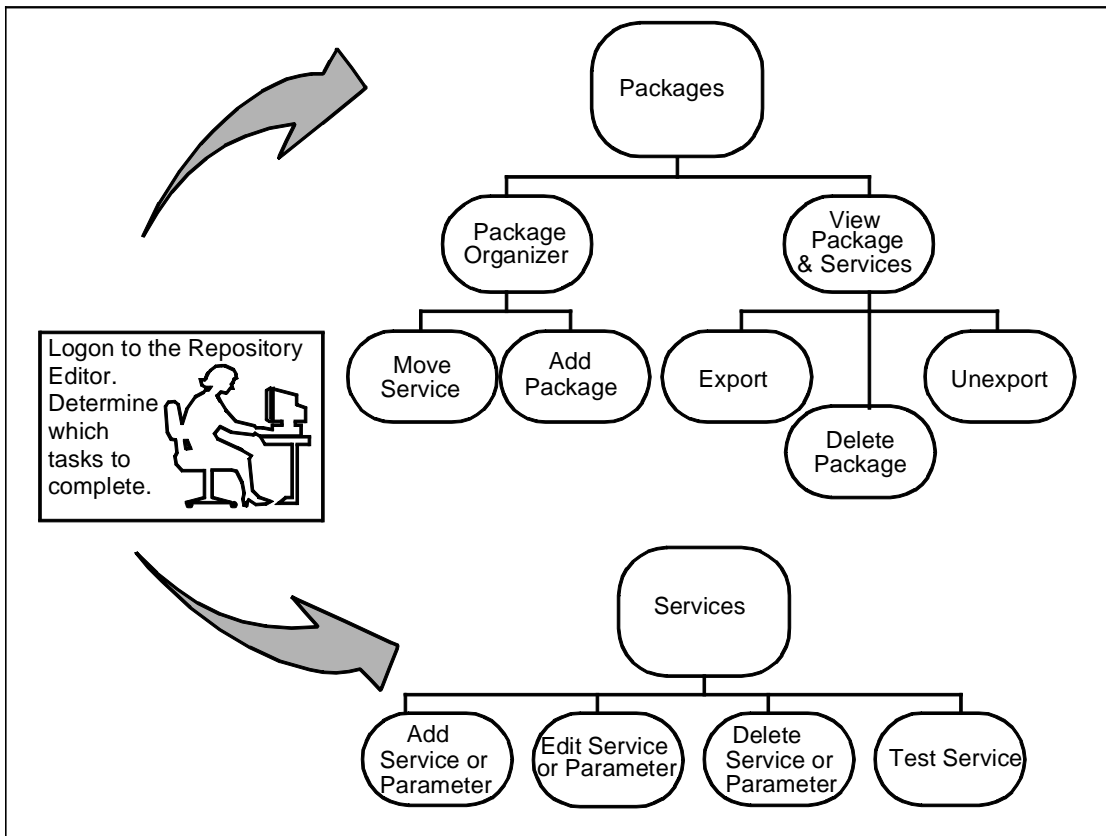
- ◆ Packages
- ◆ Services
- ◆ Parameters

In addition, you can test and group Services.

## Repository Editor Flow

After logging on to the Repository Editor, two options are enabled, **Packages** and **Services**. Figure 5-4 illustrates the Repository Editor flow to help you determine which button to select. Figure 5-4 shows the Repository Editor option flow.

Figure 5-4 Repository Editor Flow Diagram



Select **Packages** to 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

Select **Services** to access the Services window and perform the following functions:

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

## What is a Package?

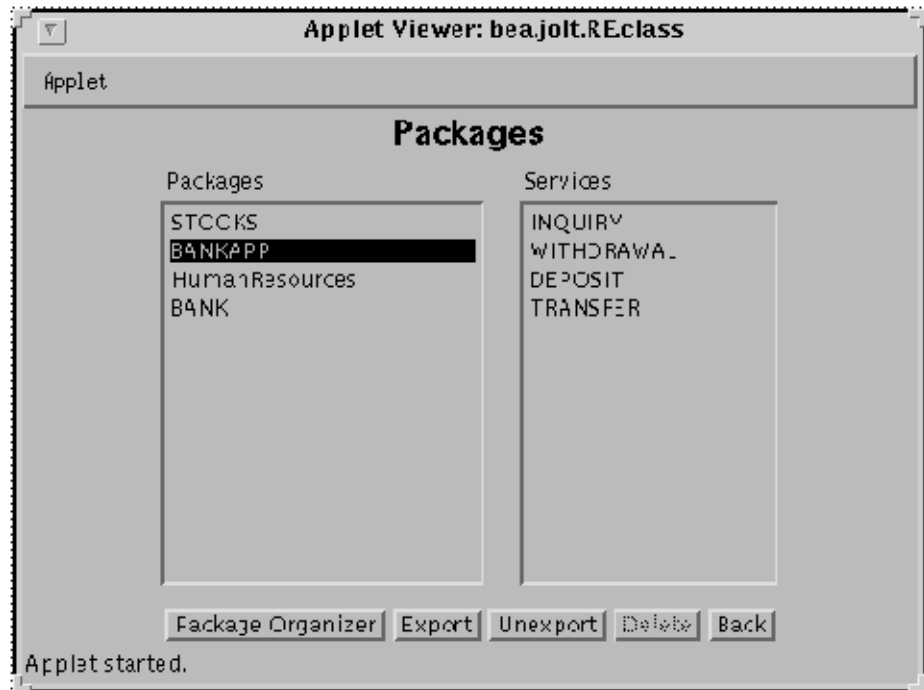
Packages provide a convenient method for grouping services for Jolt administration. A service is comprised of parameters, including pin number, account number, payment, rate, term, age, or Social Security number. The **Packages** button can be used to:

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

The available packages are displayed. When a package is selected, the services contained within a package display.

Figure 5-5 is an example of a Packages window.

**Figure 5-5 Highlighted Package with Services**



## Packages Window Description

The following listing describes the Packages window options.

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. Moves the services among the packages or add a new package.
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).

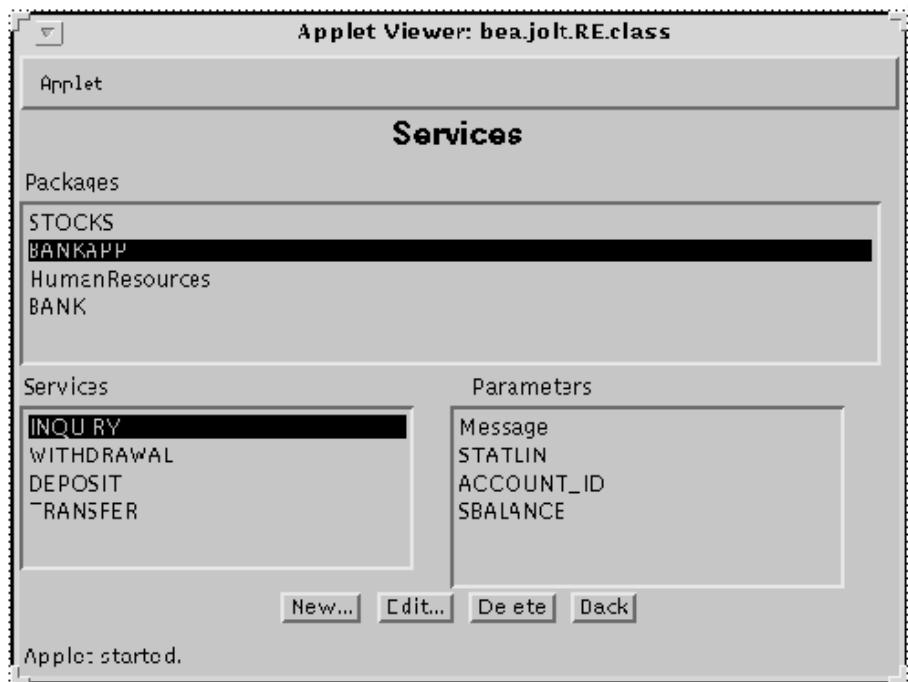
## Viewing a Package Instructions

1. To view the packages, select **Packages** from the Logon window. The Packages window displays.
2. The packages are displayed in the Packages display list. In Figure 5-5, STOCKS, BANKAPP, HumanResources, and BANK are the available packages.

## What is a Service?

A service is a definition of an available TUXEDO service. Services are comprised of parameters such as pin number, account number, payment, and rate. Adding or editing a Jolt service does not affect an existing TUXEDO service. Use the Services window to add, edit, or delete services. Figure 5-6 is an example of a Services window with the available services.

**Figure 5-6 Services Window**



## Services Window Description

The following listing describes the Services window options:

Option	Description
Packages	Lists the services and parameters for the select package. Select the package to add a new service, edit, or delete a service.
Services	Lists a service in the package to edit or delete. Selecting a service displays the parameters within the service.
Parameters	Displays selected service parameters.
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.

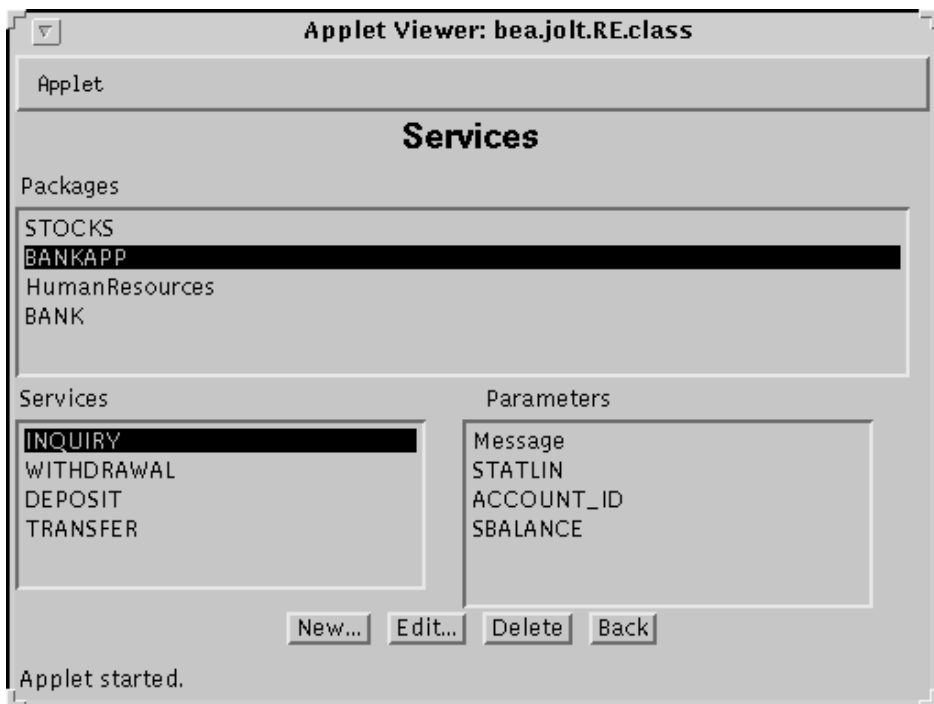
## Viewing a Service Instructions

1. To view the services, select **Services** from the Logon window. The Services window displays.
2. The available packages are displayed in the Packages display list. Selecting a package displays the available services for that package in the Services display list. In Figure 5-6, BANKAPP is the selected package.
3. The available services for the selected package are displayed in the Services display list. In Figure 5-6, INQUIRY, WITHDRAWAL, DEPOSIT, and TRANSFER are the available services for BANKAPP.

## What is a Parameter?

A service is comprised of parameters, including a pin number, account number, payment, rate, term, age, or Social Security number. A parameter is one of the service components. Adding or editing a parameter does not modify or change an existing TUXEDO service. Figure 5-7 is an example of the Services window displaying a selected service and its parameters.

**Figure 5-7 Services Window with Parameters**





### Viewing a Parameter Instructions

1. To view the parameters of a service, select **Services** from the Logon window. The Services window displays.
2. View packages in the Packages display list. To view the available services for each package, select the package. In Figure 5-7, BANKAPP is the selected package.
3. View services in the Services display list. To view the available parameters for each service, select a service. In Figure 5-7, INQUIRY is the selected service.
4. View parameters for a selected service in the Parameters display list. In Figure 5-7, Message, STATLIN, ACCOUNT\_ID, and SBALANCE are the available parameters for the INQUIRY service.

# Setting Up Packages and Services

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

- ◆ Adding a package
- ◆ Adding a service
- ◆ Adding a parameter

## Saving Your Work

As you are creating and editing services and parameters, it is important to regularly save information to ensure that you do not inadvertently lose any input. Selecting **Save Service** can prevent the need to re-enter information in the event of a system failure.

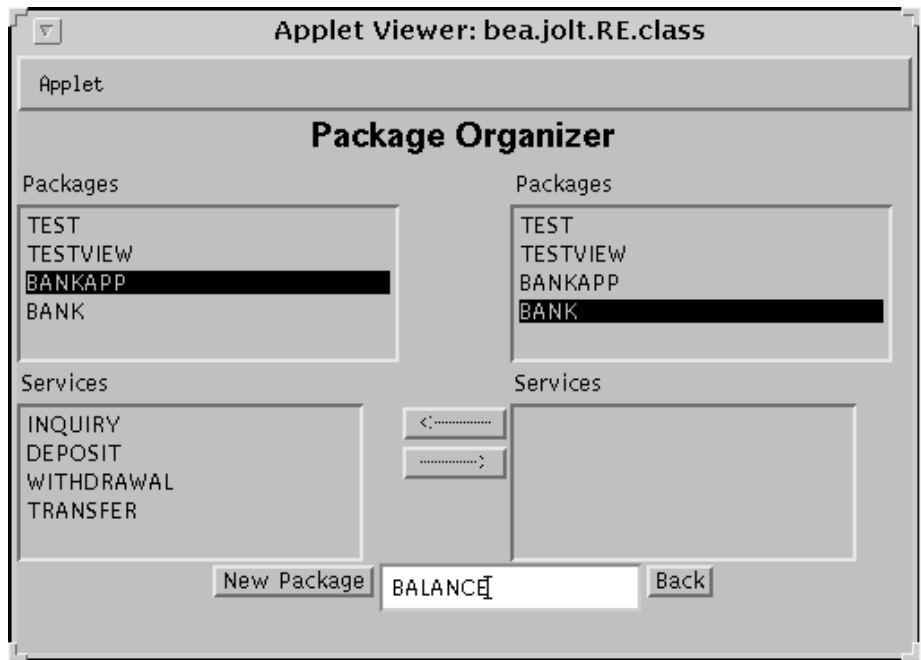
Be sure to exercise caution when you are adding or editing the parameters of a service. **Add** must be selected before choosing **Back** from the Edit Parameters window (shown in Figure 5-11) and returning to the Edit Services window (shown in Figure 5-9).

If adding a new service or modifying an existing service at the Edit Services window, ensure that **Save Service** is selected before choosing **Back**. If **Back** is selected before the modified information is saved, a warning briefly displays on the status line at the bottom of the window.

## Adding a Package

If you need to add a new group of services, a new package must be created before adding the services. Figure 5-8 shows how a new package, **BALANCE**, is added to the Packages listing.

Figure 5-8 Adding a New Package



## Adding a Package Instructions

Follow these instructions to add a package:

1. From the Logon window, select **Packages**. The Packages window displays. Select **Package Organizer**. The Package Organizer window displays. For a description of the Package Organizer window, see “Package Organizer Description” in this chapter.
2. From the Package Organizer window, select **New Package**. The text field is activated.
3. Type the name of the new package (not to exceed 32 characters) and press **Enter**. The new name (in Figure 5-8, BALANCE) is displayed in the Packages display list in random order.

## Adding a Service

Services are definitions of available TUXEDO services and can only be a part of a Jolt package. You are not required to create a new package before creating a new service; however, you must create the service as a part of a package, even if it is moved to a different package at a later date.

The Repository Editor accepts the new service name exactly as it is typed (e.g., all capital letters, abbreviations, misspellings, etc.). Service names must not exceed 30 characters. Figure 5-9 is an example of the Adding New Service window.

**Figure 5-9 Edit Services: Adding New Service Window**

**Applet viewer: bea.jolt.RE.class**

Applet

**Edit Services**

Adding new service to package: BANKAPP

Service Name

Input Buffer Type

Input View Name

Output Buffer Type

Output View Name

Parameters

Current Status: UNEXPORT

Service level actions

Parameter level actions

New Parameter...

Edit Parameter...

Delete Parameter

Save Service Test Back

## Adding a Service Window Description

The following listing describes the options for adding services to a package in a package window.

Option	Description
Service Name	Adds the name of the new service to the Repository.
Input Buffer Type/Output Buffer Type	<p>VIEW - a 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.</p> <p>VIEW32 - similar to VIEW, except 32-bit field identifiers are associated with VIEW32 structure elements.</p> <p>CARRAY - an array of uninterrupted binary data that is neither encoded nor decoded during transmission; it may contain null characters. X_OCTET is equivalent.</p> <p>FML - a type in which each field carries its own definition.</p> <p>FML32 - similar to FML except the ID field and length field are 32 bits long.</p> <p>STRING - a character array terminated by a null character that is encoded or decoded.</p>
Input View Name/Output View Name	A 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.
Current Status	Lists current status of the service. EXPORTED or UNEXPORTED status is displayed. UNEXPORTED is the default.
Save Service	Saves newly created service in the Repository.
Test	Tests the service. This is disabled until a new service is created or edits to an existing service are saved.
Parameters	Lists a parameter to edit or delete.
New Parameter	Adds new parameters to the service.
Edit Parameter	An existing parameter can be edited. This option is disabled until a new parameter is selected.
Delete Parameter	Deletes a parameter. This option is disabled until a new parameter is selected.

## Adding a Service Instructions

To add a service, follow these instructions:

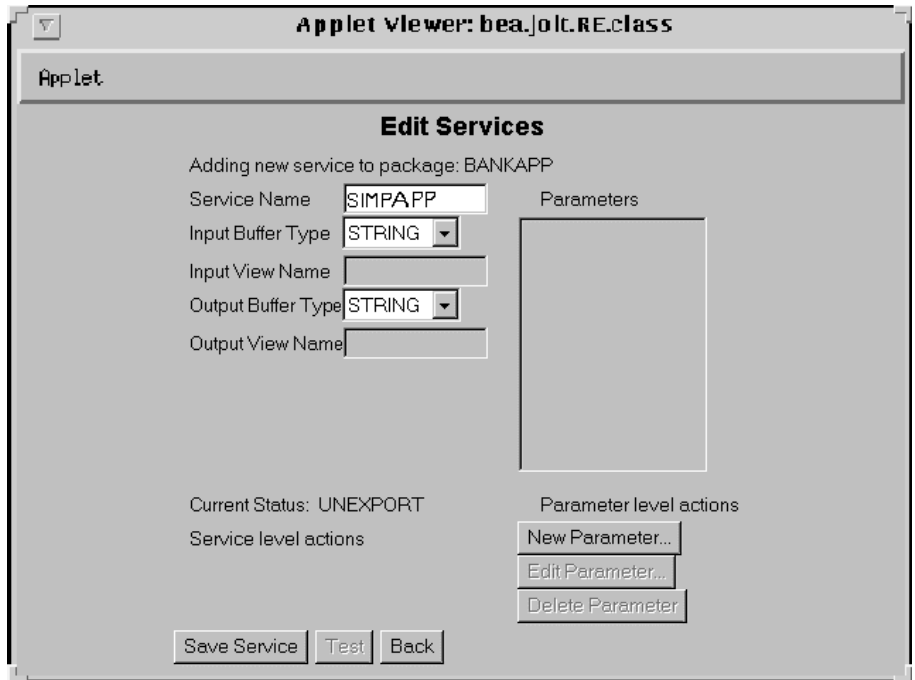
1. From the Logon window, select **Services**.
2. Select the package where the service is going to be added. If you are uncertain which package should contain the new service, select a package and use the Package Organizer to move the service to a different package. See “Grouping Services Using the Package Organizer” for additional information.
3. Select **New** from the Services window. The Edit Services window is displayed.
4. Select the **Service Name** text field to activate it. Type the service name.
5. Select the buffer type. Although the same buffer type selected for the Input Buffer is automatically selected for the Output Buffer, you can change the Output Buffer type to a different buffer type.
6. If VIEW or VIEW32 is selected, type the Input View Name and Output View Name in the accompanying text field. If another buffer type is selected, the Input View Name and Output View Name text fields are disabled. If CARRAY or STRING is selected, refer to “Selecting CARRAY or STRING as a Service Buffer Type” in this chapter 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 array or string can be added as the data type for the accompanying parameters of a service. See also “Adding a Parameter” and “Selecting array or string as a Parameter Data Type” in this chapter. For more information, refer to Chapter 6, “Using the Jolt Class Library.”

Figure 5-10 is an example of the Edit Services window with STRING as the selected buffer type for the service.

**Figure 5-10 Edit Services Window with STRING as the Selected Buffer Type**

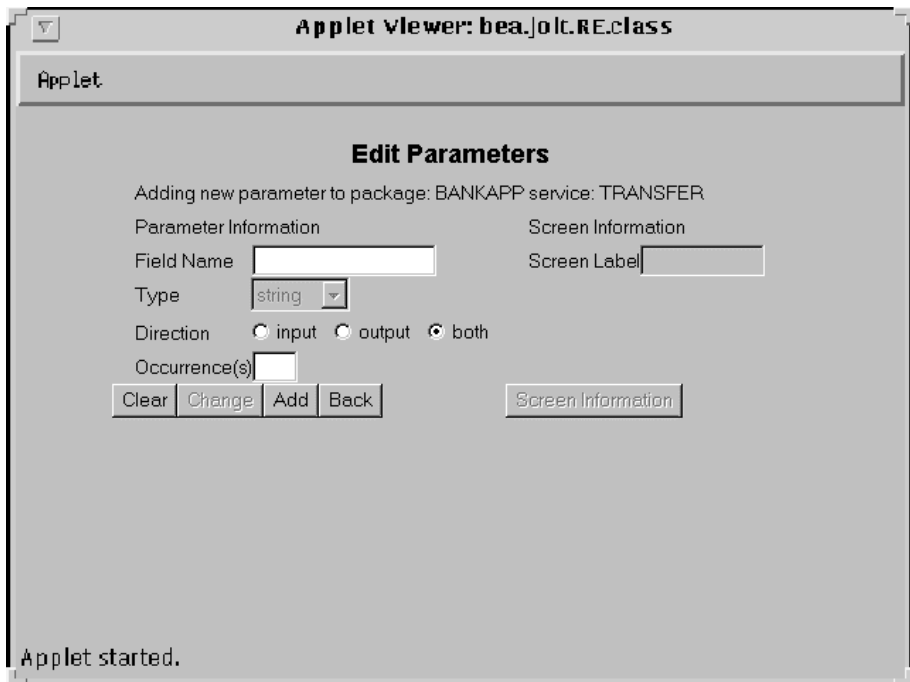


## Adding a Parameter

Selecting **New Parameter** from the Edit Services window brings up the Edit Parameters window. Review the features in Figure 5-11. Use this window to input the parameter and window information for a service.

Figure 5-11 is an example of the Edit Parameters window used to add a new parameter.

**Figure 5-11 Adding a Parameter Window**



Applet viewer: bea.jolt.RE.class

Applet

### Edit Parameters

Adding new parameter to package: BANKAPP service: TRANSFER

Parameter Information		Screen Information
Field Name	<input type="text"/>	Screen Label <input type="text"/>
Type	<input type="text" value="string"/> ▼	
Direction	<input type="radio"/> input <input type="radio"/> output <input checked="" type="radio"/> both	
Occurrence(s)	<input type="text"/>	
<input type="button" value="Clear"/> <input type="button" value="Change"/> <input type="button" value="Add"/> <input type="button" value="Back"/>		<input type="button" value="Screen Information"/>

Applet started.



## Parameters Window Description

The following listing describes the Edit Parameters window options.

Option	Description
Field Name	Adds the field name (e.g., asset, inventory).
Type	List 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	Lists choices for direction: 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	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 Jolt to build test screens; not to limit information sent or retrieved by TUXEDO.
Clear	Clears the window.
Change	Disabled while new parameters are added.
Add	Adds new parameters to the service. The parameters are saved when the service is saved.
Back	Returns user to the previous window.
Screen Label	Disabled for the Jolt 1.1 release.
Screen Information	Disabled for the Jolt 1.1 release.

## Adding a Parameter Instructions

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. Select the **Occurrences** text field to activate it, and then type the number of occurrences.
4. Specify a direction by selecting the **input**, **output**, or **both** radio buttons.
5. Select **Add** to append the information. **Add** does not save the parameter.
6. Select **Save Service** to save the parameter as a part of the service.



**Warning:** If **Save Service** is not selected before selecting **Back**, the parameters are not saved as part of the service.

7. Select **Back** to return to the previous window.

## Selecting `array` or `string` as a Parameter Data Type

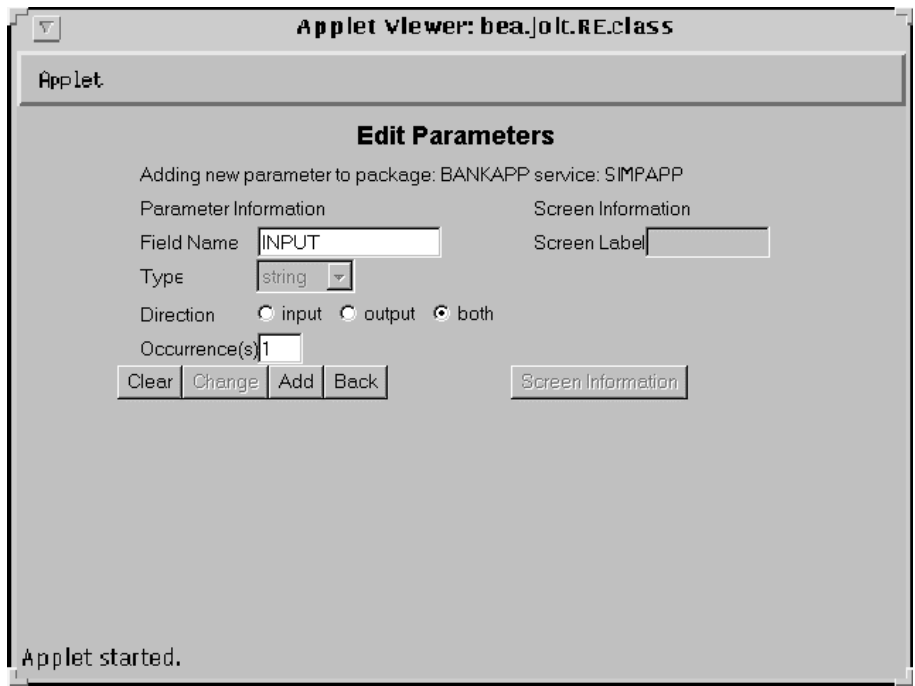
If `CARRAY` or `STRING` is the selected buffer type for a new service, only `array` or `string` can be added as the data type for the accompanying parameters of a service.

In this case, only one parameter can be added. It is recommended that the parameter name for `CARRAY` is “`CARRAY`” and the parameter name for `STRING` is “`STRING`.”

See also “Adding a Service Instructions” and “Selecting `CARRAY` or `STRING` as a Service Buffer Type” in this chapter. For more information, refer to Chapter 6, “Using the Jolt Class Library.”

Figure 5-12 is an example of the Edit Parameters window with `string` as the selected data type for the parameter. The **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 with `string` as the Data Type**



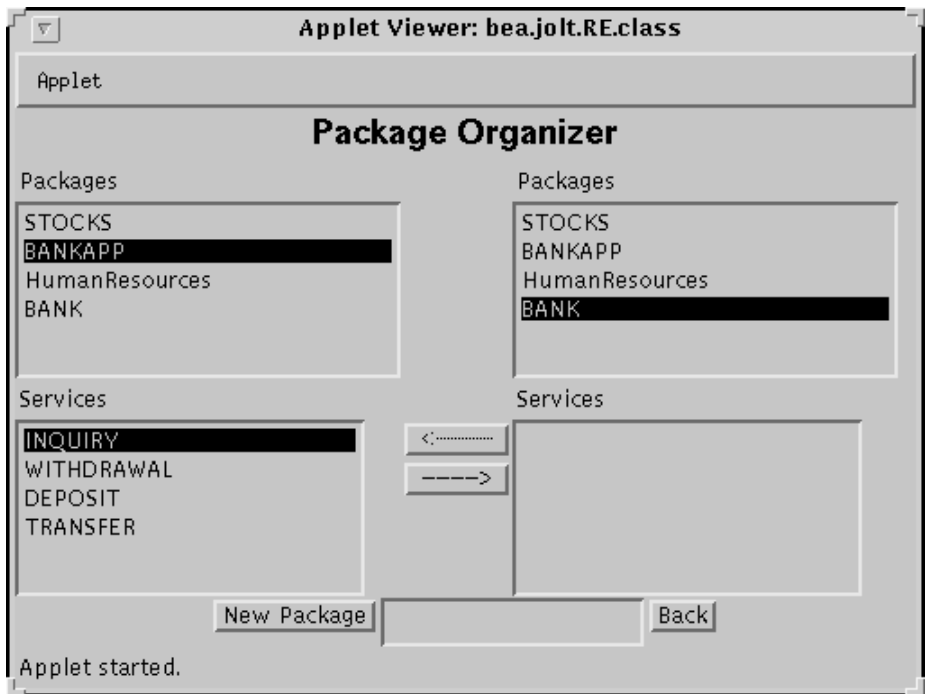
# Grouping Services Using the Package Organizer

The Package Organizer moves or transfers 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).

The Package Organizer arrow buttons allow you 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 help track a service within a particular package.

Figure 5-13 and Figure 5-14 are examples of Package Organizer windows with a service selected for transfer to another package.

**Figure 5-13** Example of a Selected Service



## Package Organizer Description

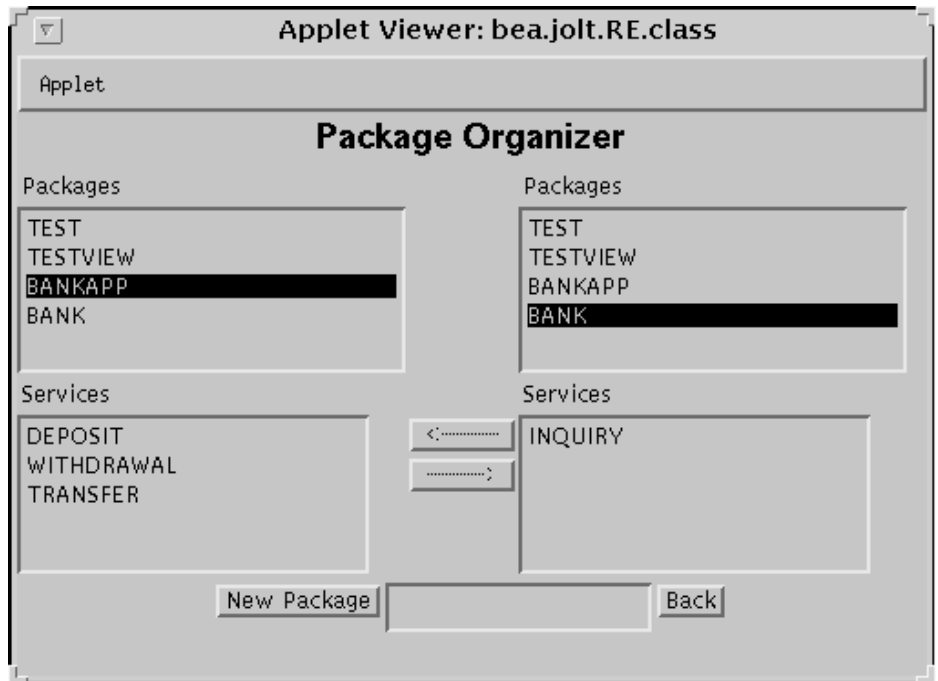
The following listing describes the options for the Package Organizer window:

Option	Description
Available Packages (left display list)	Lists packages available where the service to be moved currently resides.
Available Packages (right display list)	Lists packages available to move the service to.
Services (left display list)	Lists available services for the highlighted package that can be moved.
Services (right display list)	Lists available services that have been moved for the highlighted package.
Left arrow	Highlights services on the right to move services (one service at a time) to the package highlighted on the left.
Right arrow	Highlights services on the left to move 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 Instructions

1. Select the package containing the services to be moved from the Packages left display window to the right display window. In Figure 5-13, BANKAPP is the selected package.
2. Select the service to be moved from the Services left display window to the right display window. In Figure 5-13, INQUIRY is the selected service in the BANKAPP package.
3. Select the package to receive the service from the Packages right display window. Figure 5-13 shows the selected service and the selected package, BANK, to where the INQUIRY service will be moved.

Figure 5-14 Example of a Moved Service



4. To move the services between the packages, select the left arrow (<---) or right arrow (--->). These keys are activated only when both packages 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 how the Repository Editor moves the INQUIRY service 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/Parameters

If a package, service, or parameter requires any modifications, you can make the following changes:

- ◆ Editing a service
- ◆ Editing a parameter
- ◆ Deleting a parameter/service/package

**Note:** The Jolt 1.1 release does not allow you to edit a package name.

## Editing a Service

Edit an existing service name, service information, or access the window to add new parameters to an existing service. For a description of the Edit Services window, see “Adding a Service Window Description” in this chapter. Figure 5-15 is an example of the Edit Services window.



**Figure 5-15 Edit Services Window**

**Applet viewer: bea.jolt.RE.class**

**Applet**

**Edit Services**

Editing existing service in package: BANKAPP

Service Name:  Parameters

Input Buffer Type:

Input View Name:

Output Buffer Type:

Output View Name:

Parameters:

- FORMNAM
- SAMOUNT
- STATLIN
- ACCOUNT\_ID
- SBALANCE

Current Status: EXPORTED

Service level actions

Parameter level actions

New Parameter...

Edit Parameter...

Delete Parameter...

Save Service Test Back

## Editing a Service Instructions

Follow the instructions below to edit a service:

1. Select the package containing the service that requires editing from the Services window.
2. Select the service to edit. The parameters are displayed in the parameters display list.
3. Select **Edit**. The Edit Services window displays.
4. Type or select the new information and select **Save Service**.

## Editing a Parameter

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



**Warning:** If you are creating a new parameter using an existing name, the system overwrites the existing parameter. Figure 5-16 is an example of the Edit Parameters window.

**Figure 5-16** Edit Parameters Window

Applet viewer: bea.jolt.RE.class

Applet

### Edit Parameters

Changing existing parameter in package: BANKAPP service: TRANSFER

Parameter Information		Screen Information
Field Name	<input type="text" value="ACCOUNT_ID"/>	Screen Label <input type="text"/>
Type	<input type="text" value="integer"/>	
Direction	<input checked="" type="radio"/> input <input type="radio"/> output <input type="radio"/> both	
Occurrence(s)	<input type="text" value="2"/>	
<input type="button" value="Clear"/> <input type="button" value="Change"/> <input type="button" value="Add"/> <input type="button" value="Back"/>		<input type="button" value="Screen Information"/>

Applet started.

### Editing a Parameter Instructions

To change a parameter, follow the instructions below:

1. Select the parameter in the Parameters window and select **Edit Parameters**. The Edit Parameters: Changing Existing Parameter window displays.
2. Type the new information and select **Change**.
3. Select **Back** to return to the previous window.

### Deleting Parameters/Services/Packages

This section details the necessary sequential steps to delete a package. Before deleting a package, all of 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

Determine which parameters to delete and follow the instructions below.

1. To delete the parameters, highlight the parameter in the Parameters display list and select **Delete Parameter**.
2. Select **Back** to return to the previous window.

## Deleting a Service

Determine which services to delete and follow these instructions. Make sure that all parameters within this service are deleted before selecting this option.

1. Select **Services** from the Logon window. The Packages window displays.
2. Select the package containing the service you want to delete.
3. Select the service you want to delete. **Delete** is enabled.
4. Select **Delete**. The service is deleted.

## Deleting a Package

Determine which packages to delete and follow these instructions. Make sure all services contained in this package are deleted or moved to another package before selecting this option.

1. To delete packages, select **Packages** from the Logon window. The Packages window displays.
2. Select a package.
3. Select **Delete**. The package is deleted.

# Making a Service Available to the Jolt Client

To make a service available to a Jolt client, you must 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** buttons.

This section discusses:

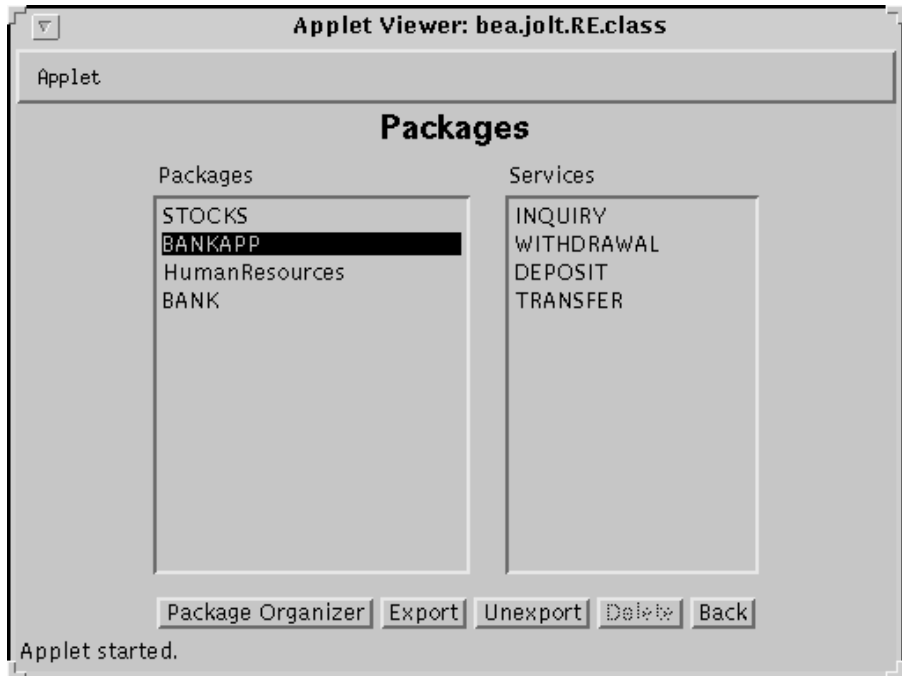
- ◆ Exporting/Unexporting services
- ◆ Reviewing the Export/Unexport status

## Exporting/Unexporting Services

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

Figure 5-17 shows the Packages window. From there you can **Export** and **Unexport** services.

**Figure 5-17 Export and Unexport Buttons**



## Exporting/Unexporting a Service Instructions

Follow the instructions below to export or unexport a service.

1. Select **Packages** from the Logon window. The Packages window displays.
2. Select a package. **Export** and **Unexport** are enabled.
3. To make services available, select **Export**.
4. To make services unavailable, select **Unexport**.

**Note:** The system does not display a confirmation message indicating that the service is exported or unexported. See “Reviewing the Exported/Unexported Status” in this chapter for additional information.

## Reviewing the Exported/Unexported Status

When a service is exported or unexported, you can review its status from the Edit Services window. Figure 5-18 shows the current status as EXPORTED.

**Figure 5-18** Exported/Unexported Status

**Applet viewer: bea.jolt.RE.class**

**Applet**

**Edit Services**

Editing existing service in package: BANKAPP

Service Name:

Input Buffer Type:

Input View Name:

Output Buffer Type:

Output View Name:

Parameters:

- FORMNAM
- SAMOUNT
- STATLIN
- ACCOUNT\_ID
- SBALANCE

Current Status: EXPORTED

Service level actions:

Parameter level actions:

- New Parameter...
- Edit Parameter...
- Delete Parameter...

Save Service Test Back

## Reviewing the Exported/Unexported Status Instructions

To review the current exported or unexported status of a service, follow these instructions:

1. Select **Services** from the Logon window. The Services window displays.
2. When you want to find out if a service has been exported or unexported, you can check its status by selecting a package from the Package display list. The Services display list is enabled with a listing of services for the selected package.
3. Select the desired service.
4. Select **Edit**. The Edit Services window displays with the **Current Status** of the service as EXPORTED or UNEXPORTED.



# Testing a Service

A service and its parameters should be tested to ensure that they are functioning properly before they are made available to Jolt clients. Services that are currently available can be tested without making changes to the services and parameters.

**Note:** The Repository Editor allows you to test an existing TUXEDO service with Jolt 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.

This section explains the following:

- ◆ Jolt Repository Editor Service Test Window
- ◆ Testing a Service Instructions

## Repository Editor Service Test Window

Test the service to ensure that the parameter information is accurate. Although **Test** is enabled when parameters are not added to the service, the Service Test window (Figure 5-19) displays the parameter fields as “unused” and they are disabled. A service can only be tested when the corresponding TUXEDO server is running for the service being tested.

**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 writable and read-only text fields.

**Figure 5-19 Sample Service Test Window**

The screenshot shows a Java Applet Viewer window titled "Applet Viewer: bea.jolt.RE.class". Inside, there's a section for "Applet" and a "Service: INQUIRY" test window. A status bar indicates "Params 1-4 of 4 displayed".

Parameter Name	Value	Type
FORMNAM		String (ReadOnly)
STATLIN		String (ReadOnly)
ACCOUNT_ID	1	integer[32]
SBALANCE		String (ReadOnly)
Unused	Unused	
Unused	Unused	
Unused	Unused	
Unused	Unused	
Unused	Unused	
Unused	Unused	

At the bottom of the window are five buttons: RUN, Clear, Next, Prev, and Back.

## Service Test Window Description

The following listing details the Service Test window 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.

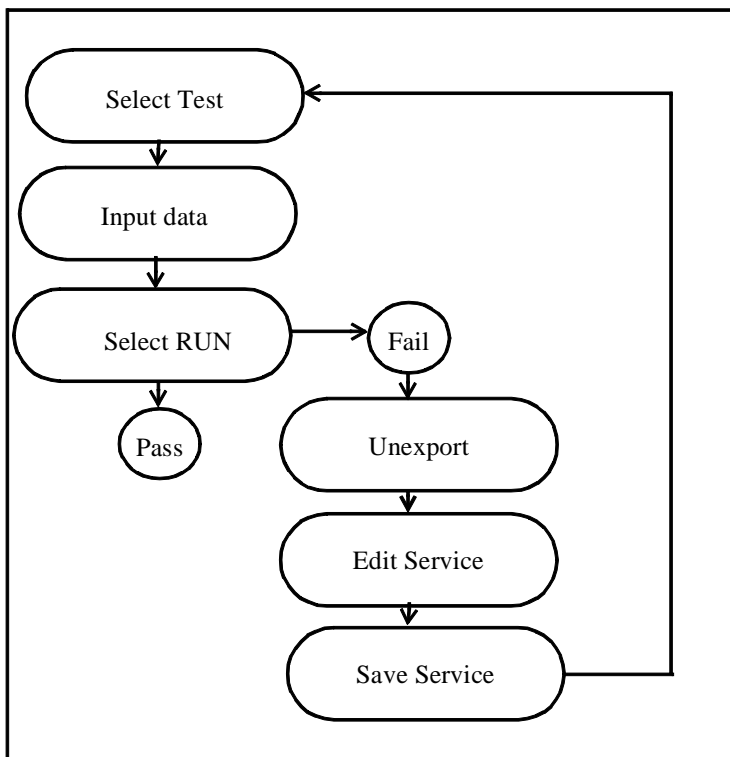
Option	Description
Service	Displays the name of the tested service (read-only).
Parameters displayed	Tracks the parameters displayed in the window (read-only).
Parameter text fields	The parameter information text entry field. These fields are writable or read-only. Disabled if read-only.
RUN	Runs the 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 Process Flow

Test a service to ensure that all service and parameter information is correct. 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.

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

**Figure 5-20 Test Service Flow**



## Testing a Service Instructions

Follow these instructions to test a service.

1. Select **Services** from the Logon window to display the Services window.
2. Select the package and the service to test.
3. Select **Edit** to access the Edit Services window.
4. Select **Test** to access the Service test window.
5. Input data in the Service test window parameter text field.
6. Select **RUN**. The status line displays the message, “Run Completed OK,” if the test passes, or “Call Failed,” if the test fails. See “Some Reasons Why a Test Might Fail” or Table 5-2 for additional Repository Editor troubleshooting information.

Follow the instructions below if editing is required to pass the test.

1. Return to the Repository Editor logon window and select **Packages**.
2. Select the package with the services to be retested.
3. Select **Unexport**.
4. Select **Back** to return to the Logon window.
5. Select **Services** to display the Services window.
6. Select the package and the service that requires editing and select **Edit**.
7. Edit the service.
8. Save the service, select **Test**, and repeat steps 5 and 6 from previous list.

## SOME REASONS WHY A TEST MIGHT FAIL

Here are some reasons why a service test might fail and possible solutions.

<b>If this . . .</b>	<b>Do this . . .</b>
A parameter is incorrect.	Edit the service.
The Jolt server is down.	Check the server. The TUXEDO service is down. You do not need to edit the service.

# Troubleshooting

If you encounter problems while using the Repository Editor, see Table 5-2.

**Table 5-2 Repository Editor Troubleshooting Table**

If . . .	Then . . .
You receive any error	<p>Make sure the browser you are running is Java-enabled:</p> <ul style="list-style-type: none"> <li>◆ For Netscape browsers, look under the “Options” menu, there should be a choice for “Show Java Console.” If this does not exist, the browser probably does not support Java.</li> <li>◆ For Internet Explorer, make sure the version is 3.0 (or later).</li> <li>◆ If running Netscape Navigator, check the Java Console for error messages.</li> <li>◆ If running appletviewer, check the system console (or the window where you started the appletviewer).</li> </ul>
You cannot connect to the Jolt Server (after entering <b>Server</b> and <b>Port Number</b> )	<p>Check and make sure that:</p> <ul style="list-style-type: none"> <li>◆ Your Server name is correct (and accessible from your machine). Check that the port number is the correct port. There must be a JSL or JRLY configured to listen on that port.</li> <li>◆ The Jolt server is up and running. If any authentication is enabled, check that you are entering the correct user names and passwords.</li> <li>◆ If the applet was loaded via http, the Web server, JRLY and the Jolt server must be on the same machine (i.e., the Server name entered into the Repository Editor must be the same machine as the one used in the URL to download the applet).</li> </ul>

**Table 5-2 Repository Editor Troubleshooting Table**

<b>If ...</b>	<b>Then ...</b>
You cannot start the Repository Editor	<p>If you are running the editor in a browser and downloading the applet via http, make sure that:</p> <ul style="list-style-type: none"> <li>◆ The browser is Java-enabled.</li> <li>◆ The Web server is running and accessible.</li> <li>◆ The <code>RE.html</code> file is available to the Web server.</li> <li>◆ The <code>RE.html</code> file contains the correct <code>&lt;codebase&gt;</code> parameter (this is where the Jolt class files are located).</li> </ul> <p>If running the editor in a browser (or appletviewer) and loading the applet from disk, make sure that:</p> <ul style="list-style-type: none"> <li>◆ The browser is Java-enabled.</li> <li>◆ The <code>RE.html</code> file exists and is readable.</li> <li>◆ The <code>RE.html</code> file is Java-enabled.</li> <li>◆ The <code>RE.html</code> file contains the correct <code>&lt;codebase&gt;</code> parameter (this is where the Jolt class files are installed on the local disk).</li> <li>◆ <code>CLASSPATH</code> is set and points to the Jolt class directory.</li> </ul>
Cannot display <b>Packages</b> or <b>Services</b> even though you are sure they exist	<ul style="list-style-type: none"> <li>◆ Make sure that the Jolt Repository Server is running (JREPSVR).</li> <li>◆ Make sure that the JREPSVR can access the repository file.</li> <li>◆ Make sure that the configuration of JREPSVR: verify CLOPT parameters and verify that <code>jrep.fl6</code> (FML definition file) is installed and accessible (follow installation documentation)</li> </ul>
Cannot save changes in the Repository Editor	Check permissions on the repository file. The file must be writable by the user who starts JREPSVR.



**Table 5-2 Repository Editor Troubleshooting Table**

<b>If ...</b>	<b>Then ...</b>
Cannot test services	<ul style="list-style-type: none"><li>◆ Check that the service is available.</li><li>◆ Verify the service definition matches the service.</li><li>◆ If TUXEDO authentication is enabled, check that you have the required permissions to execute the service.</li><li>◆ 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 is unable to process data conversion and you will receive the message "ServiceException: TPEJOLT data conversion failed."</li><li>◆ Check the ULOG file for any additional diagnostic messages.</li></ul>



# 6 Using the Jolt Class Library

The BEA Jolt Class Library provides developers with a set of new object-oriented Java language classes for accessing BEA TUXEDO services. These classes allow you to extend applications for Internet and intranet transaction processing. The application developer can use the Jolt Class Library to customize access to BEA TUXEDO services from Java applets. The following Jolt topics are included in this chapter:

- ◆ Class Library Functionality Overview
- ◆ Jolt Object Relationships
- ◆ Jolt Class Functionality
- ◆ Jolt Class Library Walk-through
- ◆ Using TUXEDO Buffer Types with Jolt
- ◆ Multithreaded Applications
- ◆ Event Subscription and Notifications
- ◆ Clearing Parameter Values
- ◆ Reusing Objects
- ◆ Application Deployment and Localization

To use the information in the following sections, you need to be generally familiar with the Java programming language and object-oriented programming concepts. All of the programming examples are in Java code.

**Note:** All of the program examples are only fragments used to illustrate Jolt capabilities. They are not intended to be compiled and run as provided. These program examples require additional code to be fully executable.

## Class Library Functionality Overview

The Jolt Class Library provides the TUXEDO application developer with the tools to develop client-side applications or applets that will run in a Java-enabled Web browser or as an independent Java application. The `bea.jolt` package contains the Jolt Class Library. To use the Jolt Class Library, the client program or applet must import this package. For an example of how to import the `bea.jolt` package, refer to Listing 6-1.

## Java Applications vs. Java Applets

Java programs that run in a browser are called “applets.” Applets are intended to be small, easily downloaded parts of an overall application that perform specific functions. Many popular browsers impose limitations on the capabilities of Java applets for the purpose of providing a high degree of security for the users of the browser. The following are some of the restrictions imposed on applets:

- ◆ An applet ordinarily cannot read or write files on any host system.
- ◆ An applet cannot start any program on the host (client) that is executing the applet.
- ◆ An applet can make a network connection only to the host where it originated; it cannot make other network connections, not even to the client machine.

Programming workarounds exist for most of the restrictions on Java applets. Check your browser’s web site (e.g., [www.netscape.com](http://www.netscape.com) or [www.microsoft.com](http://www.microsoft.com)) or developer documentation for specific information about the applet capabilities that the browser supports or restricts. You can also use Jolt Relay to overcome some of the network connection restrictions.

A Java application, however, is not run in the context of a browser and is not restricted in the same ways. For example, a Java application can start another application on the host machine where it is executing. While an applet relies on the windowing environment of a browser or appletviewer for much of its user interface, a Java application requires that you create your own user interface. An applet is designed to be small and highly portable. A Java application, on the other hand, can operate much like any other non-Java program. The security restrictions for applets imposed by various browsers and the scope of the two program types are the most important differences between a Java application and a Java applet.

## **Jolt Class Library Features**

The Jolt Class Library has the following characteristics:

- ◆ Features fully thread-safe classes.
- ◆ Encapsulates typical transaction functions such as logon, synchronous calling, transaction begin, commit, rollback, and logoffs as Java objects.
- ◆ Contains methods that allow you to set idle time-outs for continuous and intermittent client network connections.
- ◆ Features methods that allow a Jolt client to subscribe to and receive event-based notifications.

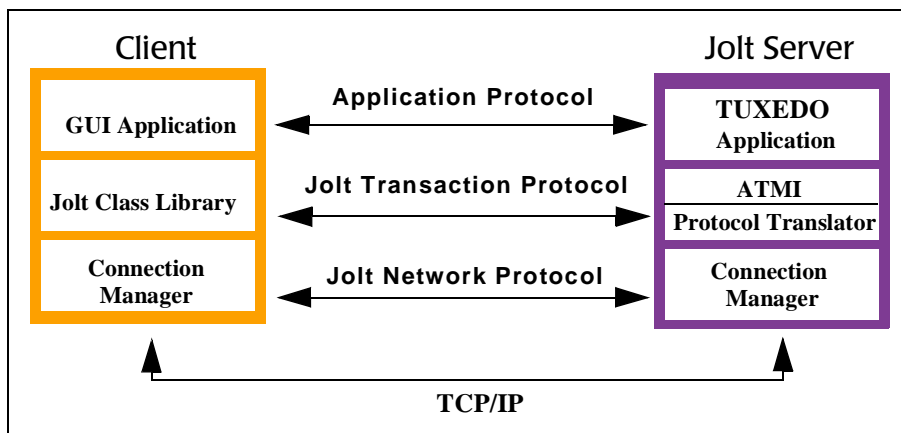
## **Jolt Class Library Error and Exception Handling**

The Jolt Class Library returns both Jolt interpreter and TUXEDO errors as exceptions. The Jolt Class Library Reference contains the Jolt classes and lists the errors or exceptions thrown for each class. Appendix A contains the Error and Exception Class Reference.

## Jolt Client/Server Relationship

BEA Jolt works in a distributed client/server environment and connects Java clients to BEA TUXEDO based applications. Figure 6-1 illustrates the client/server relationship between a Jolt program and the Jolt Server.

**Figure 6-1 Jolt Client/Server Relationship**



As illustrated in the diagram, the Jolt Server acts as a proxy for a native BEA TUXEDO client, implementing functionality available through the native BEA TUXEDO client. The BEA Jolt Server accepts requests from BEA Jolt clients and maps those requests into BEA TUXEDO service requests through the BEA TUXEDO ATMI interface. Requests and associated parameters are packaged into a message buffer and delivered over the network to the BEA Jolt Server. The BEA Jolt Connection Manager handles all communication between the BEA Jolt Server and the BEA Jolt applet using the BEA Jolt Transaction Protocol. The BEA Jolt Server unpacks the data from the message, performs any necessary data conversions, such as numeric format conversions or character set conversions, and makes the appropriate service request to BEA TUXEDO as specified by the message.

Once a service request enters the BEA TUXEDO system, it is executed in exactly the same manner as any other BEA TUXEDO request. The results are returned through the ATMI interface to the BEA Jolt Server, which packages the results and any error information into a message that is sent to the BEA Jolt client applet. The BEA Jolt client then maps the contents of the message into the various BEA Jolt client interface objects, completing the request.

On the client side, the user program contains the client application code. The Jolt Class Library packages a JoltSession and JoltTransaction, which in turn handle service requests.

The following table describes the client-side requests and Jolt Server-side actions in a simple example program.

**Table 6-1 Jolt Client/Server Interaction**

<b>Jolt Client</b>	<b>Jolt Server</b>
1 attr=new JoltSessionAttributes(); attr.setString(attr.APPADDRESS, "//myhost:8000");	Binds the client to the TUXEDO environment
2 session=new JoltSession(attr, username, userRole, userPassword, appPassword);	Logs the client on to TUXEDO
3 withdrawal=new JoltRemoteService( servname, session );	Looks up the service attributes in the Repository
4 withdrawal.addString("accountnumber", "123"); withdrawal.addFloat("amount", (float) 100.00);	Populates variables in the client (no Jolt Server activity)
5 trans=new JoltTransaction( time-out, session);	Begins a new TUXEDO transaction
6 withdrawal.call(trans);	Executes the TUXEDO service
7 trans.commit() or trans.rollback();	Completes or rolls back transaction
8 balance = withdrawal.getFloatDef("balance," (float) 0.0);	Retrieves the results (no Jolt Server activity)
9 session.endSession();	Logs the client off of TUXEDO

The following tasks summarize the interaction shown in Table 6-1 and are the steps involved in beginning a transaction:

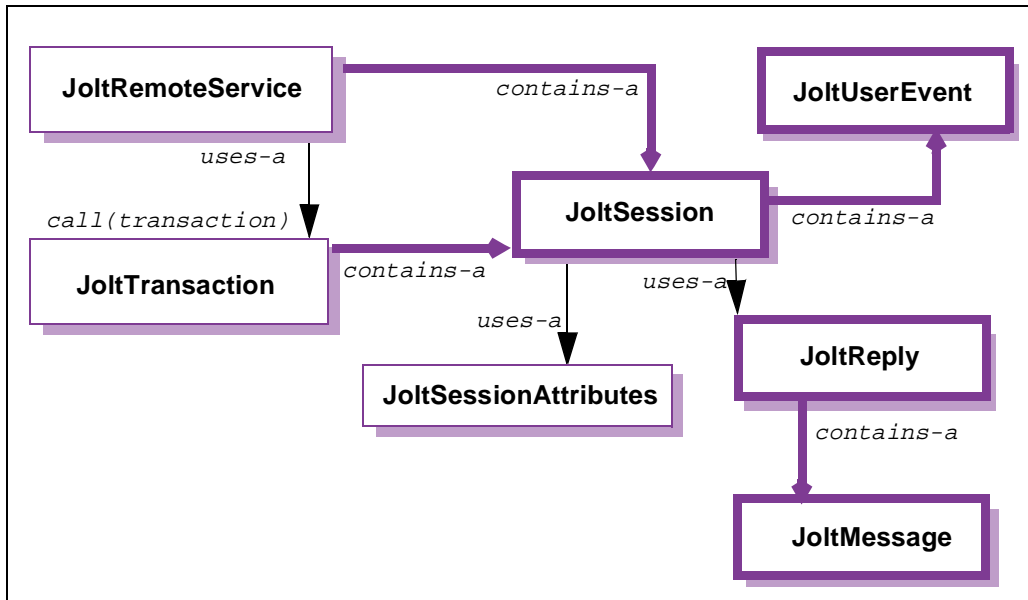
1. Bind the client to the TUXEDO environment using the `JoltSessionAttributes` class.
2. Establish a session.
3. Set variables.
4. Perform the necessary transaction processing.
5. Log the client off of the TUXEDO system.

Each of these activities is handled through the use of the Jolt Class Library classes. These classes include methods for setting and clearing data and for handling remote service actions. The following section describes the Jolt Class Library classes in more detail.

# Jolt Object Relationships

The following diagram illustrates the relationship between the instantiated objects of the Jolt Class Library classes.

**Figure 6-2 Jolt Object Relationships**



As objects, the Jolt classes interact in various relationships with each other. In Figure 6-2, the relationships are divided into three basic categories:

**Contains-a** relationship. At the class level an object can contain other objects. For example, a **JoltTransaction** stores (or contains) a **JoltSession** object.

**Is-a** relationship. The is-a relationship usually occurs at the class instance or sub-object level and denotes that the object is an instance of a particular object.

**Uses-a** relationship. An object can use another object without containing it. For example, a **JoltSession** can use the **JoltSessionAttributes** object to obtain the host and port information.



# Jolt Class Functionality

Jolt classes are used to perform the basic functions of transaction processing: log on/log off, synchronous service calling, transaction begin, commit, rollback and subscribe to events or unsolicited messages. The following sections describe how the Jolt classes are used to perform these functions.

## Logon/Logoff

The client application must log on to the TUXEDO environment prior to initiating any transaction activity. The Jolt Class Library provides the `JoltSessionAttributes` class and `JoltSession` class to establish a connection to a TUXEDO System.

The `JoltSessionAttributes` class is used to contain the connection properties to a Jolt/TUXEDO system and contains various properties about the Jolt/TUXEDO System. To establish a connection, the client application must create an instance of the `JoltSession` class. This instance is the `JoltSession` object. By instantiating a `JoltSession` object, users log on to Jolt/TUXEDO or log off by calling the `endSession` method.

## Synchronous Service Calling

Transaction activities such as requests and replies are handled through the use of a `JoltRemoteService` object (an instance of the `JoltRemoteService` class). Each `JoltRemoteService` object refers to an exported TUXEDO request/reply service. The programmer must provide a service name and a `JoltSession` object to instantiate a `JoltRemoteService` object before it can be used.

To use a `JoltRemoteService` object, the programmer simply:

- ◆ Sets the input parameters
- ◆ Invokes the service
- ◆ Examines the output parameters

For efficiency, Jolt does not make a copy of any input parameter object; only the references to the object (e.g., string and byte array) are saved. Since JoltRemoteService object is a stateful object, its input parameters and the request attributes are retained throughout the life of the object. You can use the clear() method to reset the attributes and input parameters, before reusing the JoltRemoteService object.

Since Jolt is designed for a multithreaded environment, multiple JoltRemoteService objects can be invoked simultaneously by using Java's multithreading capability. Refer to "Multithreaded Applications" in this chapter for more information.

## **Transaction Begin, Commit, and Rollback**

In Jolt, a transaction is represented as an object of the class JoltTransaction. The transaction begins when the transaction object is instantiated. The transaction object is created with a time out and JoltSession object parameter:

```
trans = new JoltTransaction(timeout, session)
```

Jolt uses an explicit transaction model for any services involved in a transaction. The transaction service invocation requires a JoltTransaction object as a parameter. Jolt also requires that the service and the transaction belong to the same session. Jolt does **not** allow you to use services and transactions that are not bound to the same session.

## Jolt Class Library Walk-through

The example code provided in Listing 6-1 shows how to use the Jolt Class Library and includes the use of the `JoltSessionAttributes`, `JoltSession`, `JoltRemoteService`, and `JoltTransaction` classes.

The example combines two user-defined TUXEDO services (WITHDRAWAL and DEPOSIT) to perform a simulated TRANSFER transaction. If the WITHDRAWAL operation fails, a rollback is performed. Otherwise, a DEPOSIT is performed and a commit completes the transaction.

The basic steps of the transaction process shown in the example are as follows:

1. Set the connection attributes like *hostname* and *portnumber* in the `JoltSessionAttribute` object. Refer to the following line in code Listing 6-1:

```
sattr = new JoltSessionAttributes();
```

2. The `sattr.checkAuthenticationLevel()` allows the application to determine the level of security required to log on to the server. Refer to the following line in code Listing 6-1:

```
switch (sattr.checkAuthenticationLevel())
```

3. The logon is accomplished by instantiating a `JoltSession` object. Refer to the following lines in code Listing 6-1:

```
session = new JoltSession (sattr, userName, userRole,  
userPassword, appPassword);
```

This example does not explicitly catch `SessionException` errors.

4. All `JoltRemoteService` calls require a service to be specified and the session key returned from `JoltSession()`. Refer to the following lines in code Listing 6-1:

```
withdrawal = new JoltRemoteService("WITHDRAWAL", session);  
  
deposit = new JoltRemoteService("DEPOSIT", session);
```

These calls bind the service definition of both the WITHDRAWAL and DEPOSIT services, which are stored in the Jolt Repository, to the withdrawal and deposit objects, respectively. The services WITHDRAWAL and DEPOSIT must be defined in the Jolt Repository otherwise a `ServiceException` will be thrown. This example does not explicitly catch `ServiceException` errors.

5. Once the service definitions are returned, the application-specific fields such as account number `ACCOUNT_ID` and withdrawal amount `SAMOUNT` are automatically populated. Refer to the following lines in code Listing 6-1:

```
withdrawal.addInt("ACCOUNT_ID", 100000);  
withdrawal.addString("SAMOUNT", "100.00");
```

The `add*()` methods can throw `IllegalAccessError` or `NoSuchFieldError` exceptions.

6. The `JoltTransaction` call allows a timeout to be specified if the transaction does not complete within the specified time. Refer to the following line in code Listing 6-1:

```
trans = new JoltTransaction(5,session);
```

7. Once the withdrawal service definition has been automatically populated, the withdrawal service is invoked by calling the `withdrawal.call(trans)` method. Refer to the following line in code Listing 6-1:

```
withdrawal.call(trans);
```

8. A failed `WITHDRAWAL` can be rolled back. Refer to the following line in code Listing 6-1:

```
trans.rollback();
```

9. Otherwise, once the `DEPOSIT` is performed, all the transactions are committed. Refer to the following lines in code Listing 6-1:

```
deposit.call(trans);  
trans.commit();
```

Listing 6-1 shows an example of a simple application for the transfer of funds using the Jolt classes.

**Listing 6-1   Jolt Transfer of Funds Example (SimXfer.java)**

---

```
/* Copyright 1996 BEA Systems, Inc.  All Rights Reserved */
import bea.jolt.*;
public class SimXfer
{
    public static void main (String[] args)
    {
        JoltSession session;
        JoltSessionAttributes sattr;
        JoltRemoteService withdrawal;
        JoltRemoteService deposit;
        JoltTransaction trans;
        String userName=null;
        String userPassword=null;
        String appPassword=null;
        String userRole="myapp";

        sattr = new JoltSessionAttributes();
        sattr.setString(sattr.APPADDRESS, "//bluefish:8501");

        switch (sattr.checkAuthenticationLevel())
        {
            case JoltSessionAttributes.NOAUTH:
                System.out.println("NOAUTH\n");
                break;
            case JoltSessionAttributes.APPASSWORD:
                appPassword = "appPassword";
                break;
            case JoltSessionAttributes.USRPASSWORD:
                userName = "myname";
                userPassword = "mysecret";
                appPassword = "appPassword";
                break;
        }
        sattr.setInt(sattr.IDLETIMEOUT, 300);
        session = new JoltSession(sattr, userName, userRole,
            userPassword, appPassword);
        // Simulate a transfer
        withdrawal = new JoltRemoteService("WITHDRAWAL", session);
        deposit = new JoltRemoteService("DEPOSIT", session);
```

```
withdrawal.addInt("ACCOUNT_ID", 100000);
withdrawal.addString("SAMOUNT", "100.00");

// Begin the transaction w/ a 5 sec timeout
trans = new JoltTransaction(5, session);
try
{
    withdrawal.call(trans);
}

catch (ApplicationException e)
{
    e.printStackTrace();
    // This service uses the STATLIN field to report errors
    // back to the client application.
    System.err.println(withdrawal.getStringDef("STATLIN", "NO
STATLIN"));
    System.exit(1);
}

String wbal = withdrawal.getStringDef("SBALANCE", "$-1.0");

// remove leading "$" before converting string to float
float w = Float.valueOf(wbal.substring(1)).floatValue();
if (w < 0.0)
{
    System.err.println("Insufficient funds");
    trans.rollback();
    System.exit(1);
}
else // now attempt to deposit/transfer the funds
{
    deposit.addInt("ACCOUNT_ID", 100001);
    deposit.addString("SAMOUNT", "100.00");

    deposit.call(trans);
    String dbal = deposit.getStringDef("SBALANCE", "-1.0");
    trans.commit();

    System.out.println("Successful withdrawal");
    System.out.println("New balance is: " + wbal);
}
```

```
        System.out.println("Successful deposit");
        System.out.println("New balance is: " + dbal);
    }

    session.endSession();
    System.exit(0);
} // end main
} // end SimXfer
```

---

## Using TUXEDO Buffer Types with Jolt

Jolt supports all of the TUXEDO typed buffers, data types, and buffer types including the built-in TUXEDO buffer types such as FML, VIEW, CARRAY, and STRING. For information about all of the TUXEDO typed buffers, data types, and buffer types, refer to the *TUXEDO System Programmer's Guide, Volume 1* and the *TUXEDO System Reference Manual*.

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

For example, if a 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 TUXEDO service. If the TUXEDO service is run on a machine whose processor uses little-endian byte-ordering (e.g., Intel processors), the TUXEDO service must convert the data properly before the data can be used.

**Note:** You should only define one parameter for the CARRAY buffer type and the STRING buffer type.

For more information about the TUXEDO CARRAY and STRING buffer types, refer *TUXEDO System Programmer's Guide, Volume 1*.

## Using the **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 Jolt to **STRING**, the null terminator is automatically appended to the end of the **STRING** buffers because Java string is not null-terminated.

The following ToUpper application fragment illustrates how Jolt works with a service whose buffer type is **STRING**. The **TOUPPER TUXEDO Service** is available in the `simpapp` example. Before running the `ToUpper.java` example in Listing 6-2, you need to do the following:

- ◆ Define the **TOUPPER** service through Jolt's Repository Editor.
- ◆ Define the **TOUPPER** service with an input buffer type of **STRING** and an output buffer type of **STRING**.
- ◆ Define an input-output parameter whose name is **STRING** for the **TOUPPER** service.
- ◆ Define only one parameter for the **TOUPPER** service.

### **Listing 6-2 Use of the **STRING** buffer type (ToUpper.java)**

---

```
/* Copyright 1996 BEA Systems, Inc. All Rights Reserved */
import bea.jolt.*;
public class ToUpper
{
    public static void main (String[] args)
    {
        JoltSession          session;
        JoltSessionAttributes sattr;
        JoltRemoteService     toupper;
        JoltTransaction       trans;
        String userName=null;
        String userPassword=null;
        String appPassword=null;
```



```
String userRole="myapp";
String outstr;

sattr = new JoltSessionAttributes();
sattr.setString(sattr.APPADDRESS, "//bluefish:8501");

switch (sattr.checkAuthenticationLevel())
{
case JoltSessionAttributes.NOAUTH:
    break;
case JoltSessionAttributes.APPPASSWORD:
    appPassword = "appPassword";
    break;
case JoltSessionAttributes.USRPASSWORD:
    userName = "myname";
    userPassword = "mysecret";
    appPassword = "appPassword";
    break;
}
sattr.setInt(sattr.IDLETIMEOUT, 300);
session = new JoltSession(sattr, userName, userRole,
userPassword, appPassword);
toupper = new JoltRemoteService ("TOUPPER", session);
toupper.setString("STRING", "hello world");
toupper.call(null);
outstr = toupper.getStringDef("STRING", null);
if (outstr != null)
    System.out.println(outstr);

session.endSession();
    System.exit(0);
} // end main
} // end ToUpper
```

---

## Using the CARRAY Buffer Type

The CARRAY buffer type is a simple character array buffer type that is built into the TUXEDO system. With the CARRAY buffer type, because the system does not interpret the data, although the data type is known, there is no way of determining how much data to transmit during an operation. The application is always required to specify a length when passing this buffer type.

The Listing 6-3 code fragment illustrates how Jolt works with a service whose buffer type is CARRAY. Since Jolt does not look into the CARRAY data stream, it is the programmer's responsibility to have the matching data format between the Jolt client and the CARRAY service.

Before running the example in Listing 6-3, you must write an “ECHO” TUXEDO service and boot the service. This service takes a buffer and passes it back. You will also need to use the Jolt Repository Editor to add in the ECHO service.

In the Repository Editor add the ECHO service as follows:

- ◆ Add a service named ECHO whose buffer type is CARRAY.
- ◆ Define the ECHO service with an input-output parameter named CARRAY.
- ◆ Define just one parameter for the CARRAY buffer type.

### Listing 6-3 Use of CARRAY Buffer Type

---

```
/* Copyright 1996 BEA Systems, Inc. All Rights Reserved */

/*
 * This code fragment illustrates how Jolt works with a service whose
 * buffer type is CARRAY.
 */

import java.io.*;

import bea.jolt.*;

class ...
{
    ...
    public void tryOnCARRAY()
    {
        byte data[];
```

```

JoltRemoteService csvc;
DataInputStream din;
DataOutputStream dout;
ByteArrayInputStream bin;
ByteArrayOutputStream bout;
/*
 * Use java.io.DataOutputStream to put data into a byte array
 */
bout = new ByteArrayOutputStream(512);
dout = new DataOutputStream(bout);
dout.writeInt(100);
dout.writeFloat((float) 300.00);
dout.writeUTF("Hello World");
dout.writeShort((short) 88);
/*
 * Copy the byte array into a new byte array "data". Then
 * issue the Jolt remote service call.
 */
data = bout.toByteArray();
csvc = new JoltRemoteService("ECHO", session);
csvc.setBytes("CARRAY", data, data.length);
csvc.call(null);
/*
 * Get the result from JoltRemoteService object and use
 * java.io.DataInputStream to extract each individual value
 * from the byte array.
 */
data = csvc.getBytesDef("CARRAY", null);
if (data != null)
{
    bin = new ByteArrayInputStream(data);
    din = new DataInputStream(bin);
    System.out.println(din.readInt());
    System.out.println(din.readFloat());
    System.out.println(din.readUTF());
    System.out.println(din.readShort());
}
}
}

```

---

## Using the VIEW Buffer Type

For Jolt 1.1, information about using the TUXEDO View buffer type is located at the following BEA web site address:

<http://www.beasys.com/products/jolt/index.htm>

## Using the FML Buffer Type

The Listing 6-4 Java code fragment illustrates how Jolt works with a service whose buffer type is FML.

### Listing 6-4 Use of the FML Buffer Type

---

```

/* Copyright 1997 BEA Systems, Inc. All Rights Reserved */
/*
 * This code fragment illustrates how Jolt works with a service whose
 * buffer type is FML.
 */
import bea.jolt.*;
public class /* start of ... */ tryOnFml
{
    public static void main (String [] args)
    {
        tryOnFml();
    }
    public static void tryOnFml ()
    {
        JoltSessionAttributes sattr = null;
        JoltRemoteService passFml;
        JoltSession session;
        String outputString;
        int outputInt;
        float outputFloat;

        sattr = new JoltSessionAttributes();
        sattr.setString(sattr.APPADDRESS, "//bluefish:5151");
        session = new JoltSession(sattr, "test", "role", null, null);
        /* end of ... */
        passFml = new JoltRemoteService("PASSFML",session);
        passFml.setString("INPUTSTRING", "John");
    }
}

```

```
passFml.setInt("INPUTINT", 67);
passFml.setFloat("INPUTFLOAT", (float)12.0);
passFml.call(null);
outputString = passFml.getStringDef("OUTPUTSTRING", "failed");
outputInt = passFml.getIntDef("OUTPUTINT", -1);
outputFloat = passFml.getFloatDef("OUTPUTFLOAT", (float)-1.0);
System.out.print("String =" + outputString);
System.out.print(" Int =" + outputInt);
System.out.println(" Float =" + outputFloat);
}
}
```

---

**Note:** The example `tryOnFml.c` illustrates the server side code for using the FML buffer type. This example can be found at the following BEA Web site address:

<http://www.beasys.com/products/jolt/index.htm>

## FML Field Definitions

The following entries show FML field definitions for the `tryOnFml.java` example.

```
#
# FML field definition table
#
*base      4100
INPUTSTRING 1    string
INPUTINT    2    long
INPUTFLOAT  3    float
OUTPUTSTRING 4    string
OUTPUTINT   5    long
OUTPUTFLOAT 6    float
```

# Multithreaded Applications

As a Java based set of classes, Jolt supports multithreaded applications. However, various implementations of the Java language differ with respect to certain language and environment features. Jolt programmers need to be aware of the following:

- ◆ The use of preemptive and non-preemptive threads when creating applications or applets with the Jolt Class Library.
- ◆ The use of threads to get asynchronous behavior similar to the `tpacall()` function in TUXEDO.

The following section describes the issues arising from using threads with different Java implementations and is followed by an example of the use of threads in a Jolt program.

## Preemptive and Non-preemptive Threads

Most Java implementations provide preemptive threads. However, the current Sun Solaris implementation provides non-preemptive threads. The difference between these two models can lead to very different performance and programming requirements.

## Threads of Control

Each concurrently operating task in the Java virtual machine is a thread. Threads exist in various states, the important ones being **RUNNING**, **RUNNABLE**, or **BLOCKED**.

- ◆ A **RUNNING** thread is a currently executing thread.
- ◆ A **RUNNABLE** thread can be run once the current thread has relinquished control of the CPU. There can be many threads in the **RUNNABLE** state, but only one can be in the **RUNNING** state. Running a thread means changing the state of a thread from **RUNNABLE** to **RUNNING**, and causing the thread to have control of the Java Virtual Machine (VM).
- ◆ A **BLOCKED** thread is a thread that is waiting on the availability of some event or resource.

**Note:** The Java VM schedules threads of the same priority to run in a round-robin mode.

## Preemptive Threading

The main performance difference between the two threading models arises in telling a running thread to relinquish control of the Java VM. In a preemptive threading environment, the usual procedure is to set a hardware timer that goes off periodically; when the timer goes off, the current thread is moved from the **RUNNING** to the **RUNNABLE** state, and another thread is chosen to run.

## Non-preemptive Threading

In a non-preemptive threading environment, a thread must volunteer to give up control of the CPU and move to the **RUNNABLE** state. Many of the methods in the Java language classes contain code that volunteers to give up control, and are typically associated with actions that might take a long time. For instance, reading from the network will generally cause a thread to wait for a packet to arrive. A thread that is waiting on the availability of some event or resource is in the **BLOCKED** state. When the event occurs or the resource becomes available, the thread becomes **RUNNABLE**.

## Using Jolt with Non-Preemptive Threading

If your Jolt-based Java program is running on a non-preemptive threading Virtual Machine (e.g., Sun Solaris), then the program must either:

- ◆ Occasionally call a method that blocks the thread
- ◆ Explicitly give up control of the CPU using the `Thread.yield()` method

The typical usage is to make the following call in all long running code segments or potentially time-consuming loops:

```
Thread.currentThread().yield();
```

Without sending this message, the threads used by the Jolt library may never get scheduled, and as such, the Jolt operation will be impaired.

The only virtual machine known to use non-preemptive threading is the Java Developer's Kit (JDK version 1.0, 1.0.1, 1.0.2) machine running on a Sun platform. If you want your applet to work on JDK 1.0, you must make sure to send the yield messages. As mentioned earlier, some methods contain yields. An important exception is the `System.in.read` method. This method does not cause a thread switch. Rather than rely on these messages, we suggest using yields explicitly.

**Note:** Sun has indicated that JDK 1.1 will implement preemptive threads, and should alleviate the requirement for yields. Code that includes yields will continue to work; code without yields will begin working with the JDK 1.1 release.

## Using Threads for Asynchronous Behavior

You can use threads in Jolt to get asynchronous behavior that is analogous to the `tpacall()` function in TUXEDO. With this capability, you do not need a asynchronous service request function. You can get this functionality because Jolt is thread safe. For example the client Jolt application can start one thread that sends a request to a TUXEDO service function and then immediately starts another thread that sends another request to a TUXEDO service function. So even though the Jolt `tpacall()` is synchronous, the application is asynchronous because the two threads are running at the same time.

## Using Threads with Jolt

A Jolt client-side program or applet is fully thread-safe. Jolt's support of multi-threaded applications includes the following client characteristics:

- ◆ Multiple sessions per client
- ◆ Multithreaded within a session
- ◆ Client application manages threads, not asynchronous calls
- ◆ Performs synchronous calls

The following program illustrates the use of two threads in a Jolt application.



**Listing 6-5   Using Multithreads with Jolt (ThreadBank.java)**

---

```
/* Copyright 1996 BEA Systems, Inc.  All Rights Reserved */
import bea.jolt.*;
public class ThreadBank
{
    public static void main (String [] args)
    {
        JoltSession session;
        try
        {
            JoltSessionAttributes dattr;
            String userName = null;
            String userPasswd = null;
            String appPasswd = null;
            String userRole = null;

            // fill in attributes required
            dattr = new JoltSessionAttributes();
            dattr.setString(dattr.APPADDRESS,"//bluefish:8501");

            // instantiate domain
            // check authentication level
            switch (dattr.checkAuthenticationLevel())
            {

            case JoltSessionAttributes.NOAUTH:
                System.out.println("NOAUTH\n");
                break;
            case JoltSessionAttributes.APPASSWORD:
                appPasswd = "myAppPasswd";
                break;
            case JoltSessionAttributes.USRPASSWORD:
                userName = "myName";
                userPasswd = "mySecret";
                appPasswd = "myAppPasswd";
                break;

            }

            dattr.setInt(dattr.IDLETIMEOUT, 60);
            session = new JoltSession (dattr, userName, userRole,
                                     userPasswd, appPasswd);
```

```
T1 t1 = new T1 (session);
T2 t2 = new T2 (session);

t1.start();
t2.start();

Thread.currentThread().yield();
try
{
    while (t1.isAlive() && t2.isAlive())
    {
        Thread.currentThread().sleep(1000);
    }
}
catch (InterruptedException e)
{
    System.err.println(e);
    if (t2.isAlive())
    {
        System.out.println("job 2 is still alive");
        try
        {
            Thread.currentThread().sleep(1000);
        }
        catch (InterruptedException e1)
        {
            System.err.println(e1);
        }
    }
    else if (t1.isAlive())
    {
        System.out.println("job1 is still alive");
        try
        {
            Thread.currentThread().sleep(1000);
        }
        catch (InterruptedException e1)
        {
            System.err.println(e1);
        }
    }
}
session.endSession();
}
catch (SessionException e)
```

```

        {
            System.err.println(e);
        }
        finally
        {
            System.out.println("normal ThreadBank term");
        }
    }
}

class T1 extends Thread
{
    JoltSession j_session;
    JoltRemoteService j_withdrawal;

    public T1 (JoltSession session)
    {
        j_session=session;
        j_withdrawal= new JoltRemoteService("WITHDRAWAL",j_session);
    }

    public void run()
    {
        j_withdrawal.addInt("ACCOUNT_ID",10001);
        j_withdrawal.addString("SAMOUNT","100.00");
        try
        {
            System.out.println("Initiating Withdrawal from account 10001");
            j_withdrawal.call(null);
            String W = j_withdrawal.getStringDef("SBALANCE","-1.0");
            System.out.println("-->Withdrawal Balance: " + W);
        }
        catch (ApplicationException e)
        {
            e.printStackTrace();
            System.err.println(e);
        }
    }
}

class T2 extends Thread
{
    JoltSession j_session;
    JoltRemoteService j_deposit;

    public T2 (JoltSession session)
    {

```

```
        j_session=session;
        j_deposit= new JoltRemoteService("DEPOSIT",j_session);
    }
    public void run()
    {
        j_deposit.addInt("ACCOUNT_ID",10000);
        j_deposit.addString("SAMOUNT","100.00");
        try
        {
            System.out.println("Initiating Deposit from account 10000");
            j_deposit.call(null);
            String D = j_deposit.getStringDef("SBALANCE","-1.0");
            System.out.println("-->Deposit Balance: " + D);
        }
        catch (ApplicationException e)
        {
            e.printStackTrace();
            System.err.println(e);
        }
    }
}
```

---

# Event Subscription and Notifications

Programmers developing client applications using Jolt can receive event notifications from either TUXEDO Services or other TUXEDO clients. The Jolt Class Library contains classes that support the following types of TUXEDO notifications for handling event-based communication:

- ◆ **Unsolicited Event Notifications.** These are notifications that a Jolt client receives as a result of a TUXEDO client or service issuing a broadcast using either a *tpbroadcast()* or a directly targeted message via a *tpnotify()* ATMI call.
- ◆ **Brokered Event Notifications.** These notifications are received by a Jolt client via the TUXEDO Event Broker. The notifications are only received when both the Jolt client subscribes to an event and any TUXEDO client or server issues system posted event or a *tppost()* call.

## API for Event Subscription

The Jolt Class Library provides four classes that implement the asynchronous notification mechanism for Jolt client applications. These classes are:

- ◆ **JoltSession.** The JoltSession class includes an *onReply()* method for receiving notifications and notification messages.
- ◆ **JoltReply.** The JoltReply class gives the client application access to any messages received with an event or notification.
- ◆ **JoltMessage.** The JoltMessage class provides get methods for obtaining information about the notification or event.
- ◆ **JoltUserEvent.** The JoltUserEvent class supports subscription to of both unsolicited and event notification types.

For more information about these classes refer to Chapter 7, “Jolt Class Library Reference.”

## Notification Event Handler

For both unsolicited notifications and a brokered event notification, the Jolt client application requires an event handler routine that is invoked upon receipt of a notification. The Jolt 1.1 release only supports a single handler per session. In current TUXEDO versions, it is not possible to determine which event generated a notification. Thus it is not possible to invoke an event-specific handler based on a particular event.

The client application must provide a single handler (by overriding the `onReply()` method) per session that will be invoked for all notifications received by that client for that session. The single handler call-back function is used for both unsolicited and event notification types. It is up to the (user supplied) handler routine to determine what event caused the handler invocation and take appropriate action. If the user does not override the session handler, then notification messages are silently discarded by the default handler.

The Jolt client provides the call back function by subclassing the `JoltSession` class and overriding the `onReply()` method with a user-defined `onReply()` method.

In TUXEDO/ATMI clients, processing in the handler call-back function is limited to a subset of ATMI calls. This restriction does not apply to Jolt clients. Separate threads are used to monitor notifications and run the event handler method. A Jolt client will be able to perform all Jolt-supported functionality from within the handler. All the rules that apply to a normal Jolt client program apply to the handler, such as a single transaction per session at any time.

Each invocation of the handler method takes place in a separate thread. The application developer should ensure that the `onReply()` method is either synchronized or written thread-safe, since separate threads could be executing the method simultaneously.

## Subscribing to Event Notifications Enables Unsolicited Notifications

Jolt uses an implicit model for enabling the handler routing. When a client subscribes to an event, Jolt will internally enable the handler for that client, thus enabling unsolicited notifications as well. A Jolt client cannot subscribe to event notifications without also receiving unsolicited notifications. In addition, a single `onReply()` method is invoked for both types of notifications.

## Connection Modes

Jolt supports notification receipt for clients working in either connection-retained or connection-less modes of operation. Connection-retained clients receive all notifications. Jolt clients working in connection-less mode will receive notifications while they have an active network connection to the Jolt Session Handler (JSH). When the network connection is closed, the JSH logs and drops notifications destined for the client. Jolt clients operating in a connection-less mode will not receive unsolicited messages or notifications while they do not have an active network connection. All messages received during this time are logged and discarded by the JSH.

### ACKNOWLEDGED NOTIFICATIONS

Connection mode notification handling includes acknowledged notifications for Jolt 1.1 clients in a TUXEDO 6.3 environment. If a JSH receives an acknowledged notification for a client and the client does not have an active network connection, the JSH logs an error and return a failure acknowledgment to the notification.

## Notification Data Buffers

When a client receives notification, it is accompanied by a data buffer. The data buffer can be of any TUXEDO data buffer type. Jolt clients (i.e., the handler) will receive these buffers as a `JoltMessage` object and should use the appropriate `JoltMessage` class `get*()` methods to retrieve the data from this object.

The Jolt Repository does not need to have the definition of the buffers used for notification. However, the Jolt client application programmer will need to know field names beforehand.

The Jolt system does not provide functionality equivalent to `tptypes()` in TUXEDO, so in effect a Jolt 1.1 client is limited to receiving a “known” buffer type. For FML and VIEW buffers, the data will be accessed using the `get*()` methods with the appropriate field name, for example, enter:

```
getIntDef ("ACCOUNT_ID", -1);
```

For **STRING** and **CARRAY** buffers, the data will be accessed by the same name as the buffer type, e.g.,:

```
getStringDef ("STRING", null);  
getBytesDef ("CARRAY", null);
```

**STRING** and **CARRAY** buffers contain only a single data element: this complete element is returned in the `get*( )` methods above.

## **TUXEDO Event Subscription**

TUXEDO brokered event notification allows TUXEDO programs to post events without needing to know what other programs are supposed to receive notification of an event's occurrence. The Jolt event notification allows Jolt client applications to subscribe to TUXEDO events that are broadcast or posted using the TUXEDO `tpnotify()` or `tpbroadcast()` calls.

Jolt clients are only able to subscribe to events and notifications that are generated by other components in TUXEDO (such as a TUXEDO Service or Client). Jolt 1.1 clients are not able to send events or notifications.

### **Supported Subscription Types**

Jolt1.1 only supports notification types of subscriptions. The Jolt `onReply()` method is called when a subscription is fulfilled. The Jolt 1.1 API does not support dispatching a service routine or enqueueing a message to an application queue when a notification is received.

### **Subscribing to Notifications**

If a Jolt client subscribes to a single event notifications, the client receives both unsolicited messages and event notification. Subscribing to an event implicitly enables unsolicited notification. This means that if the application creates a `JoltUserEvent` object for Event "X", the client will automatically receive notifications directed to it as a result of `tpnotify()` or `tpbroadcast()`.



**Note:** This is NOT the recommended method for enabling unsolicited notification - if you want unsolicited notification, the application should explicitly do so (as described in the `JoltUserEvent` class). The reason for this is explained in the following unsubscribe section.

## Unsubscribing from Notifications

To stop subscribing to event notifications and/or unsolicited messages, you need to use the `JoltUserEvent` unsubscribe method. In Jolt, disabling unsolicited notifications with an unsubscribe method does not turn off all subscription notifications. This differs from TUXEDO. In TUXEDO the use of `tpsetunsol()` with a NULL handler turns off all subscription notifications.

When unsubscribing, the following considerations apply:

- ◆ If a client is subscribed to only a single event, unsubscribing disables both the event notification and unsolicited messages.
- ◆ If a client has multiple subscriptions, then unsubscribing from any single subscription disables just that single subscription. Unsolicited notifications continue. Only the last subscription to be unsubscribed causes unsolicited notification to stop.
- ◆ If a client subscribes to both an unsolicited and an event notifications, then unsubscribing to just the unsolicited notification will not stop either type of notifications from continuing. In addition, this unsubscribe does not throw an Exception. However, the Jolt API will remember that an unsubscribe has taken place and a subsequent unsubscribe to the remaining event will disable both event notification and unsolicited messages.

If you want to stop unsolicited messages in your client application, you need to make sure that you have unsubscribed to all events.

**Note:** Jolt 1.1 does not support the wildcard unsubscribe semantics of `tpunsubscribe(-1, . .)`. Jolt clients wishing to unsubscribe to all subscriptions will do so by invoking the `unsubscribe()` method in each event subscription object.

## Using the Jolt API to Receive TUXEDO Notifications

The example code provided in Listing 6-6 shows how to use the Jolt Class Library for receiving notifications and includes the use of the `JoltSession`, `JoltReply`, `JoltMessage` and `JoltUserEvent` classes.

---

### Listing 6-6 Asynchronous Notification

---

```
class EventSession extends JoltSession
{
    public EventSession( JoltSessionAttributes attr, String user,
                        String role, String upass, String apass )
    {
        super(attr, user, role, upass, apass);
    }
    /**
     * Override the default unsolicited message handler.
     * @param reply a place holder for the unsolicited message
     * @see bea.jolt.JoltReply
     */
    public void onReply( JoltReply reply )
    {
        // Print out the STRING buffer type message which contains
        // only one field; the field name must be "STRING".  If the
        // message uses CARRAY buffer type, the field name must be
        // "CARRAY".  Otherwise, the field names must conform to the
        // elements in FML or VIEW.

        JoltMessage msg = (JoltMessage) reply.getMessage();
        System.out.println(msg.getStringDef("STRING", "No Msg"));
    }
    public static void main( Strings args[] )
    {
        JoltUserEvent    unsolEvent;
        JoltUserEvent    helloEvent;
        EventSession     session;
        ...

        // Instantiate my session object which can print out the
        // unsolicited messages.  Then subscribe to HELLO event
```

```
// and Unsolicited Notification which both use STRING
// buffer type for the unsolicited messages.

session = new EventSession(...);

helloEvent = new JoltUserEvent("HELLO", null, session);
unsolEvent = new JoltUserEvent(JoltUserEvent.UNSOLMSG, null,
                               session);

...
// Unsubscribe the HELLO event and unsolicited notification.
helloEvent.unsubscribe();
unsolEvent.unsubscribe();
    }
}
```

---

# Clearing Parameter Values

The Jolt Class Library includes a method (the `clear` method) that allows the developer to remove an object's existing attributes and in effect provides for the reuse of the object. The `reuseSample.java` example illustrates how to use the `clear` method for clearing parameter values.

The `reuseSample.java` example shows how to reuse the `JoltRemoteService` parameter values. The example shows that you do not have to destroy the service to reuse it. Instead, the `svc.clear();` statement is used to discard the existing input parameters before reusing the `addString` method.

## Listing 6-7 Jolt Object Reuse (`reuseSample.java`)

---

```
/* Copyright 1996 BEA Systems, Inc. All Rights Reserved */
import java.net.*;
import java.io.*;
import bea.jolt.*;
/*
 * This is a Jolt sample program that illustrates how to reuse the
 * JoltRemoteService after each invocation.
 */
class reuseSample
{
    private static JoltSession s_session;
    static void init( String host, short port )
    {
        /* Prepare to connect to the TUXEDO domain. */
        JoltSessionAttributes attr = new JoltSessionAttributes();
        attr.setString(attr.APPADDRESS,"//"+ host+": " + port);

        String username = null;
        String userrole = "sw-developer";
        String applpasswd = null;
        String userpasswd = null;

        /* Check what authentication level has been set. */
        switch (attr.checkAuthenticationLevel())
        {
```

```
        case JoltSessionAttributes.NOAUTH:
            break;
        case JoltSessionAttributes.APPASSWORD:
            applpasswd = "secret8";
            break;
        case JoltSessionAttributes.USRPASSWORD:
            username = "myName";
            userpasswd = "BEA#1";
            applpasswd = "secret8";
            break;
    }

    /* Logon now without any idle timeout (0). */
    /* The network connection is retained until logoff. */
    attr.setInt(attr.IDLETIMEOUT, 0);
    s_session = new JoltSession(attr, username, userrole,
        userpasswd, applpasswd);
}

public static void main( String args[] )
{
    String host;
    short  port;
    JoltRemoteService svc;

    if (args.length != 2)
    {
        System.err.println("Usage: reuseSample host port");
        System.exit(1);
    }

    /* Get the host name and port number for initialization. */
    host = args[0];
    port = (short)Integer.parseInt(args[1]);

    init(host, port);
}
```

```
/* Get the object reference to the DELREC service. This
 * service has no output parameters, but has only one input
 * parameter.
 */
svc = new JoltRemoteService("DELREC", s_session);
try
{
    /* Set input parameter REPNAME. */
    svc.addString("REPNAME", "Record1");
    svc.call(null);
    /* Change the input parameter before reusing it */
    svc.setString("REPNAME", "Record2");
    svc.call(null);

    /* Simply discard all input parameters */
    svc.clear();
    svc.addString("REPNAME", "Record3");
    svc.call(null);
}
catch (ApplicationException e)
{
    System.err.println("Service DELREC failed: "+
        e.getMessage()+" "+ svc.getStringDef("MESSAGE", null));
}

/* Logoff now and get rid of the object. */
s_session.endSession();
}
}
```

---

## Reusing Objects

The following `extendSample.java` example illustrates one way to subclass the `JoltRemoteService` class. In this case, a `TransferService` class is created by subclassing the `JoltRemoteService` class. The `TransferService` class extends the `JoltRemoteService` class, adding a `Transfer` feature which makes use of the TUXEDO bankapp funds TRANSFER service.

The example uses the “extends” mechanism from the Java language. The `extend` is used in Java to subclass a base (parent) class. The following code shows only one of many different ways to extend from `JoltRemoteService`.

### **Listing 6-8 Extending Jolt Remote Service (`extendSample.java`)**

---

```
/* Copyright 1996 BEA Systems, Inc. All Rights Reserved */

import java.net.*;
import java.io.*;
import bea.jolt.*;

/*
 * This Jolt sample code fragment illustrates how to customize
 * JoltRemoteService. It uses the Java language “extends” mechanism
 */
class TransferService extends JoltRemoteService
{
    public String    fromBal;
    public String    toBal;

    public TransferService( JoltSession session )
    {
        super( "TRANSFER", session );
    }

    public String doxfer( int fromAcctNum, int toAcctNum, String
amount )
    {
        /* Clear any previous input parameters */
        this.clear();
    }
}
```

```
        /* Set the input parameters */
        this.setIntItem("ACCOUNT_ID", 0, fromAcctNum);
        this.setIntItem("ACCOUNT_ID", 1, toAcctNum);
        this.setString("SAMOUNT", amount );

    try
    {
        /* Invoke the transfer service. */
        this.call(null);

        /* Get the output parameters */
        fromBal = this.getStringItemDef("SBALANCE", 0, null);
        if (fromBal == null)
            return "No balance from Account " +
                fromAcctNum;
        toBal = this.getStringItemDef("SBALANCE", 1, null);
        if (toBal == null)
            return "No balance from Account " + toAcctNum;
        return null;
    }
    catch (ApplicationException e)
    {
        /* The transaction failed, return the reason */
        return this.getStringDef("STATLIN", "Unknown reason");
    }
}

class extendSample
{
    public static void main( String args[] )
    {
        JoltSession s_session;
        String      host;
        short       port;
        TransferService xfer;
        String      failure;

        if (args.length != 2)
        {
            System.err.println("Usage: reuseSample host port");
        }
    }
}
```



```

System.exit(1);
}

/* Get the host name and port number for initialization. */
host = args[0];
port = (short)Integer.parseInt(args[1]);

/* Prepare to connect to the TUXEDO domain. */
JoltSessionAttributes attr = new JoltSessionAttributes();
attr.setString(attr.APPADDRESS, "/" + host + ":" + port);

String username = null;
String userrole = "sw-developer";
String applpasswd = null;
String userpasswd = null;

/* Check what authentication level has been set. */
switch (attr.checkAuthenticationLevel())
{
    case JoltSessionAttributes.NOAUTH:
        break;
    case JoltSessionAttributes.APPASSWORD:
        applpasswd = "secret8";
        break;
    case JoltSessionAttributes.USRPASSWORD:
        username = "myName";
        userpasswd = "BEA#1";
        applpasswd = "secret8";
        break;
}

/* Logon now without any idle timeout (0). */
/* The network connection is retained until logoff. */
attr.setInt(attr.IDLETIMEOUT, 0);
s_session = new JoltSession(attr, username, userrole,
userpasswd, applpasswd);

/*
 * TransferService extends from JoltRemoteService and uses the
 * standard TUXEDO BankApp TRANSFER service. We invoke this
 * service twice with different parameters. Note, we assume
 * that "s_session" is initialized somewhere before.
 */

```

```
xfer = new TransferService(s_session);
if ((failure = xfer.doxfer(10000, 10001, "500.00")) != null)
    System.err.println("Tranasaction failed: " + failure);
else
{
    System.out.println("Transaction is done.");
    System.out.println("From Acct Balance: "+xfer.fromBal);
    System.out.println("  To Acct Balance: "+xfer.toBal);
}

if ((failure = xfer.doxfer(51334, 40343, "$123.25")) != null)
    System.err.println("Tranasaction failed: " + failure);
else
{
    System.out.println("Transaction is done.");
    System.out.println("From Acct Balance: "+xfer.fromBal);
    System.out.println("  To Acct Balance: "+xfer.toBal);
}
}
}
```

---

# Application Deployment and Localization

The Jolt Class Library allows you to build Java applications that execute from within a client Web browser. For these types of applications, you need to address the following application development tasks:

- ◆ Deploying your Jolt application in an HTML page
- ◆ Localizing your Jolt application for different languages and character sets

The following sections describe these application development considerations.

## Deploying a Jolt Applet

When you deploy a Jolt applet, you need to consider the three components that operate together to make the applet function in a Web browser environment:

- ◆ Requirements for the TUXEDO server and Jolt Server
- ◆ Client-side execution of the applet
- ◆ Requirements for the Web server that downloads the Java applet

Information for configuring the TUXEDO server and Jolt Server to work with Jolt is available in Chapter 2, “Installing Jolt.” The following sections describe common client and Web server considerations for deploying Jolt applets.

## Client Considerations

When you write a Java applet that incorporates Jolt classes, the applet works just as any other Java applet in an HTML page. A Jolt applet can be embedded in a HTML page using the HTML applet tag:

```
<applet code="applet_name.class"> </applet>
```

If the Jolt applet is embedded in an HTML page, the applet downloads when the HTML page loads. You can code the applet to run immediately after it is downloaded, or you can include code that sets the applet to run based upon a user action, a timeout, or a set interval. You can also create an applet that downloads in the HTML page, but opens in another window or, for instance, simply plays a series of sounds or musical tunes at intervals. The programmer has a large degree of freedom in coding the applet initialization procedure.

**Note:** If the user loads a new HTML page into the browser, the applet execution is stopped.

## Web Server Considerations

When you use the Jolt classes in a Java applet, the Jolt Server must run on the same machine as the Web server that downloads the Java applet unless you install Jolt Relay on the Web server.

When a webmaster sets up a Web server, a directory is specified to store all the HTML files. Within that directory, a subdirectory named “classes” must be created to contain all Java class files and packages. For example:

```
<html-dir>/classes/bea/jolt
```

**Note:** You can place the Jolt classes subdirectory anywhere. For convenient access, you may want to place it in the same directory as the HTML files. The only requirements for the Jolt classes subdirectory are that they must be made available to the Web server.

Since all Jolt classes belong to package `bea.jolt`, all Jolt class files are put in `/classes/bea/jolt` subdirectory (i.e., “jolt” is a subdirectory of “bea” which is a subdirectory of “classes”).

The HTML file for the Jolt applet should refer the codebase to the “classes” directory. For example:

```
/export/html/  
|___ classes/  
|___ |___ bea/  
|___ |___ |___ jolt/  
|___ |___ |___ JoltSessionAttributes.class  
|___ |___ |___ JoltRemoteServices.class  
|___ |___ |___ ...  
|___ |___ mycompany/  
|___ |___ app.class  
|___ ex1.html  
|___ ex2.html
```

The webmaster may specify the “app” applet in ex1.html as:

```
<applet codebase="classes" code=mycompany.app.class width=400  
height=200>
```

## Localizing a Jolt Applet

If your Jolt application is intended for international use, you must address certain application localization issues. Localization considerations apply to applications that execute from a client Web browser and applications that are designed to run outside a Web browser environment. Localization tasks can be divided into two categories:

- ◆ Adapting an application from its original language to a target language.
- ◆ Translating strings from one language to another. This sometimes requires specifying a different alphabet or a character set from the one used in the original language.

For localization, the Jolt Class Library package relies on the conventions of the Java language and the TUXEDO system. Jolt transfers Java 16-bit Unicode characters to the JSH. The JSH provides a mechanism to convert Unicode to the local character set.

For information about the Java implementation for Unicode and character escapes, refer to your Java Development Kit (JDK) documentation.



# 7 Jolt Class Library Reference

The Jolt Class Library consists of object-oriented Java language classes for accessing BEA TUXEDO services and defining transactions. The Jolt Class Library package is designed to be small, simple, and easy to incorporate into your Java application. The classes for this package provide TUXEDO logon/logoff, synchronous calling, and transaction services for a client Java applet.

The Jolt Class Library reference includes the following topics:

- ◆ Jolt Methods
- ◆ JoltSessionAttributes Class
- ◆ JoltSession Class
- ◆ JoltRemoteService Class
- ◆ JoltRequestMessage Abstract Class
- ◆ JoltTransaction Class
- ◆ JoltEvent Class
- ◆ JoltUserEvent Class
- ◆ JoltReply Class
- ◆ JoltMessage Class

Each class description includes information about the class constructor and class methods.

To use the following information, you need to be generally familiar with the Java programming language and object-oriented programming concepts. The reference material follows the Java standard terminology for classes and for the methods that operate on the class instances.

For information about programming the Jolt Class Library, refer to Chapter 6, “Using the Jolt Class Library.” For information about the Jolt Exception Classes and related TUXEDO errors listed for the Jolt classes, refer to Appendix A, “Jolt Class Library Errors and Exceptions.”

## **Jolt Methods**

The Jolt classes and class methods that form the Jolt Class Library follow the Java language structure and are intended for use in Java programs. However, since the Jolt Class Library is designed for accessing TUXEDO System applications, you need to be aware of how get, set, add, and delete methods operate within the context of transaction processing.

For example, if you are working with a series of account numbers for a banking transaction and are changing the value from one account number to another account number, you must be aware of how the Jolt methods determine the position of an item in a list. Understanding how these methods work will help you avoid inadvertently reversing the order of a transaction, assigning incorrect values to an item, or deleting the wrong item.

## **Methods for Handling Items**

The get, set, and setItem methods are used to obtain information about a specified item or to change information about a specified item. The add methods append a specified item to a list and the delete and deleteItem methods remove specified items. The following sections describe how these methods operate on an item in a list of items.

## Changing the First Item Occurrence

The set and delete methods operate on the first occurrence of an item in a list. In Jolt, the first occurrence of an item is always at position zero. For example, if you have a list of account numbers and you use a delete method, the item at position zero is deleted. For example:

```
delete ( "ACCTNUM" )
```

Deletes the account number at position zero of the account number list.

```
setInt ( "ACCTNUM", 3200 )
```

Sets the account number at position zero to the value 3200.

## Changing Items by Item Position

The following types of methods in the Jolt Class Library are position dependent. To change a position-dependent item, you specify the location or index of the item in a list.

- ◆ set...Item methods

- ◆ deleteItem methods

For example, to set the third item in a list of account numbers to a value, you would specify the following:

```
setIntItem ( "ACCTID", 2, 5000 )
```

The example sets the account number at position three to the value 5000. You specify an index of 2 because the index numbering starts at 0 and 2 is at the third position of the list.

To delete the first item in a list of account numbers, you would specify the following:

```
deleteItem ( "ACCTID", 0 )
```

The example deletes the account number at position 0 and is exactly equivalent to using just the delete method. A **set...Item** (*name*, 0, *value*) also works exactly as the set method.



## Getting Items

The Jolt Class Library includes a number of get methods. However, the get methods operate differently from the other types of methods. The following three get methods retrieve information with a single value.

- ◆ `getOccurrenceCount`
- ◆ `getName`
- ◆ `getApplicationcode`

All other get methods are in the form of a `get...Def` or `get...ItemDef`. The `get...Def` methods retrieve information about the first item (item 0). The `get...ItemDef` methods retrieve information about an item that you specify by its position or index number. If the item has no information, these methods allow you to set a default. For example:

```
getIntItemDef ("CreditRating", 4, -1)
```

The example gets the credit rating for the fifth item in a list of items starting at position zero. If the item has no value for a credit rating, the get returns the default value of -1. You specify the value for the method in the *defValue* parameter.

## Appending Items

The add methods append an item to the end of a list of items. Of course if your list of items is empty, an add will make the item the first one in the list. In the following example, the add method appends a name to the end of a list of names:

```
addString ("CompanyName", "BoltBikeWorks")
```

# JoltSessionAttributes Class

```
java.lang.Object
|
+----bea.jolt.JoltSessionAttributes
```

The JoltSessionAttributes class defines the acceptable attributes for the JoltSession constructor. The set methods add the attribute if one does not exist. Otherwise, the set method overwrites the old value.

The get and set methods throw the java.lang.NoSuchFieldError or java.lang.IllegalAccessException. By default, these errors are caught by the Java virtual machine.

```
public class JoltSessionAttributes
{
    /* The supported authentication levels. */
    public final static int NOAUTH;           // No authentication
    public final static int APPASSWORD;      // Application password
    public final static int USRPASSWORD;     // App and User passwords

    /* The attribute names for set and get. */
    public final static String APPADDRESS    // String "//host:port"
                                           // Used with setString()
    public final static String IDLETIMEOUT;  // Used with SetInt()"
    public final static String SENDTIMEOUT;  // Used with SetInt()"
    public final static String RECVTIMEOUT;  // Used with SetInt()"

    /* The attribute name for get only. */
    public final static String SESSIONTIMEOUT; // getIntDef()

    public JoltSessionAttributes();

    //JoltSessionAttributes methods.
    public int  checkAuthenticationLevel()
                throws SessionException;

    public void clear();
    public byte getByteDef( String name, byte defValue );
    public byte[] getBytesDef( String name, byte[] defValue );
    public double getDoubleDef( String name, double defValue );
    public float getFloatDef( String name, float defValue );
```

```
public int      getIntDef( String name, int defValue );
public short   getShortDef( String name, short defValue );
public String   getStringDef( String name, String defValue );
public void     setByte( String name, byte value );
public void     setBytes( String name, byte[] value, int len);
public void     setDouble( String name, double value );
public void     setFloat( String name, float value );
public void     setInt( String name, int value );
public void     setShort( String name, short value );
public void     setString( String name, String value );
}
```

## JoltSessionAttributes Constructor

The following constructor creates an instance of the JoltSessionAttributes class.

### JoltSessionAttributes

This constructor allocates a new instance of the JoltSessionAttributes class.

Synopsis    `public JoltSessionAttributes();`

Usage        Specific components are extracted to assist in defining the attributes of the Session constructor.

Throws       `java.lang.IllegalAccessError`. Occurs when the user tries to set a value to a get-only attribute, or to get a value from a set-only attribute.

`java.lang.NoSuchFieldError`. Occurs when the user tries to set an attribute which is not one of the predefined attributes.

## JoltSessionAttributes Methods

The following methods are used in conjunction with the JoltSessionAttributes class.

- ◆ **checkAuthenticationLevel and clear methods.** These methods provide a means to retrieve the values set for system access and to remove data.
- ◆ **get methods.** The get attribute methods obtain the value of an attribute. If the attribute does not exist, the default value is returned. The user sets the default.
- ◆ **set methods.** These methods set the value of an attribute. If a value already exists, using the set method overwrites the existing value. Unless otherwise specified, the set methods return no values.

These JoltSessionAttributes methods are described in the following sections.

### checkAuthenticationLevel

The checkAuthenticationLevel method gets the authentication level set up by the TUXEDO administrator.

**Synopsis**    `int checkAuthenticationLevel() throws SessionException;`

**Usage**    The authentication level determines which values are set to the JoltSession constructor parameters. This method returns the authentication level specified in Table 7-1.

You must set APPADDRESS before calling this method (refer to Table 7-2).

Returns Table 7-1 describes the possible return values for the `checkAuthenticationLevel` method.

**Table 7-1 Return Values for the `checkAuthenticationLevel` Method**

Return Value	Description
NOAUTH	No authentication is required to access the system.
APPASSWORD	System authentication is required. Clients provide user name, user role, and application password to authenticate the system.
USRPASSWORD	System and application authentication are required to access the system. Clients provide user name, user role, user password, and application password to authenticate the system.

Throws `java.lang.IllegalAccessError`. Occurs when the user tries to set a value to a get-only attribute, or to get a value from a set-only attribute.

`java.lang.NoSuchFieldError`. Occurs when the user tries to set an attribute which is not one of the predefined attributes.

## clear

The `clear` method removes all attributes.

Synopsis `void clear();`

Usage The `JoltSessionAttributes` class contains data that must be removed for object reuse. Use the `clear` method to remove data.

## getBytesDef

The `getBytesDef` method gets the byte value of a specified item.

**Synopsis**    `byte getBytesDef ( String name, byte defValue );`

**Usage**       Gets the byte value (8-bit) of the item specified in the *name* parameter or the specified *defValue*, if the name does not exist.

**Returns**     This method returns a single byte value.

**Throws**       `java.lang.IllegalAccessError`. Occurs when the user tries to set a value to a get-only attribute, or to get a value from a set-only attribute.

`java.lang.NoSuchFieldError`. Occurs when the user tries to set an attribute which is not one of the predefined attributes.

## getBytesDef

The `getBytesDef` method gets an array of byte values.

**Synopsis**    `byte[ ] getBytesDef ( String name, byte[] defValue );`

**Usage**       Gets an array of byte values of the item specified in the *name* parameter or the specified *defValue*, if the name does not exist.

**Returns**     This method returns an array of values.

**Throws**       `java.lang.IllegalAccessError`. Occurs when the user tries to set a value to a get-only attribute, or to get a value from a set-only attribute.

`java.lang.NoSuchFieldError`. Occurs when the user tries to set an attribute which is not one of the predefined attributes.

## getDoubleDef

The `getDoubleDef` method gets the double precision value of an item.

**Synopsis**     `double getDoubleDef( String name, double defValue );`

**Usage**     Gets the double precision value of the item specified in the *name* parameter or the specified *defValue*, if the name does not exist.

**Returns**     This method returns an double precision floating point value (64-bit).

**Throws**     `java.lang.IllegalAccessError`. Occurs when the user tries to set a value to a get-only attribute, or to get a value from a set-only attribute.

`java.lang.NoSuchFieldError`. Occurs when the user tries to set an attribute which is not one of the predefined attributes.

## getFloatDef

The `getFloatDef` method gets the floating point value of an item.

**Synopsis**     `float getFloatDef( String name, float defValue );`

**Usage**     Gets the floating point value of the item specified in the *name* parameter or the specified *defValue*, if the name does not exist.

**Returns**     This method returns a floating point value (32-bit).

**Throws**     `java.lang.IllegalAccessError`. Occurs when the user tries to set a value to a get-only attribute, or to get a value from a set-only attribute.

`java.lang.NoSuchFieldError`. Occurs when the user tries to set an attribute which is not one of the predefined attributes.

## getIntDef

The `getIntDef()` method gets the integer value of an item.

**Synopsis**     `int getIntDef( String name, int defValue );`

<b>Parameters</b>	<i>name</i>	Specifies the attribute name.
	<i>defValue</i>	Specifies a default value for an item if it does not have an existing name.

**Usage**     Gets the integer value of the item specified in the *name* parameter or the specified *defValue*, if the name does not exist. Use these values to specify the *defValue* parameter for this method.

**Returns**     This method returns an integer value (32-bit).

**Example**     `attr.getIntDef(attr.SESSIONTIMEOUT, 1);`

**Throws**     `java.lang.IllegalAccessError`. Occurs when the user tries to set a value to a get-only attribute, or to get a value from a set-only attribute.

`java.lang.NoSuchFieldError`. Occurs when the user tries to set an attribute which is not one of the predefined attributes.



## getShortDef

The `getShortDef` method gets the short integer value of an item.

**Synopsis**    `short getShortDef( String name, short defValue );`

**Usage**    Gets the short integer value of the item specified in the *name* parameter or the specified *defValue*, if the name does not exist.

**Returns**    This method returns a short integer value (16-bit).

**Throws**    `java.lang.IllegalAccessError`. Occurs when the user tries to set a value to a get-only attribute, or to get a value from a set-only attribute.

`java.lang.NoSuchFieldError`. Occurs when the user tries to set an attribute which is not one of the predefined attributes.

## getStringDef

The `getStringDef` method gets the string value of an item.

**Synopsis**    `String getStringDef( String name, String defValue );`

**Usage**    Gets the string value of the item specified in the *name* parameter or the specified *defValue*, if the name does not exist. The default value can be a null string or any other text string.

**Returns**    This method returns a text string.

**Throws**    `java.lang.IllegalAccessError`. Occurs when the user tries to set a value to a get-only attribute, or to get a value from a set-only attribute.

`java.lang.NoSuchFieldError`. Occurs when the user tries to set an attribute which is not one of the predefined attributes.

## setByte

The setByte method sets the byte value of a specified item.

**Synopsis**    `void setByte( String name, byte value );`

**Usage**       Sets the value of the item specified in the *name* parameter to the byte value specified in the *value* parameter. The value is an 8-bit byte.

**Throws**     `java.lang.IllegalAccessError`. Occurs when the user tries to set a value to a get-only attribute, or to get a value from a set-only attribute.

`java.lang.NoSuchFieldError`. Occurs when the user tries to set an attribute which is not one of the predefined attributes.

## setBytes

The setBytes method sets the byte array value of a specified item.

**Synopsis**    `void setBytes( String name, byte[ ] value, int len );`

**Usage**       Sets the value of the item specified in the *name* parameter to the byte array value specified in the *value* parameter, with the length set by the *len* parameter. The value is in 8-bit bytes.

**Throws**     `java.lang.IllegalAccessError`. Occurs when the user tries to set a value to a get-only attribute, or to get a value from a set-only attribute.

`java.lang.NoSuchFieldError`. Occurs when the user tries to set an attribute which is not one of the predefined attributes.

## setDouble

The setDouble method sets the double precision value of a specified item.

Synopsis    `void setDouble( String name, double value );`

Usage      Sets the value of the item specified in the *name* parameter to the double precision value specified in the *value* parameter. The value is in 64-bits.

Throws     `java.lang.IllegalAccessError`. Occurs when the user tries to set a value to a get-only attribute, or to get a value from a set-only attribute.

`java.lang.NoSuchFieldError`. Occurs when the user tries to set an attribute which is not one of the predefined attributes.

## setFloat

The setFloat method sets the floating point value of a specified item.

Synopsis    `void setFloat( String name, float value );`

Usage      Sets the value of the item specified in the *name* parameter to the floating point (32-bit) value specified in the *value* parameter.

Throws     `java.lang.IllegalAccessError`. Occurs when the user tries to set a value to a get-only attribute, or to get a value from a set-only attribute.

`java.lang.NoSuchFieldError`. Occurs when the user tries to set an attribute which is not one of the predefined attributes.

## setInt

The setInt method sets the integer value of a specified item.

**Synopsis**    `void setInt( String name, int value );`

**Usage**       Sets the value of the item specified in the *name* parameter to the integer (32-bit) value specified in the *value* parameter.

**Throws**      `java.lang.IllegalAccessError`. Occurs when the user tries to set a value to a get-only attribute, or to get a value from a set-only attribute.

`java.lang.NoSuchFieldError`. Occurs when the user tries to set an attribute which is not one of the predefined attributes.

## setShort

The setShort method sets the short integer value of a specified item.

**Synopsis**    `void setShort( String name, short value );`

**Usage**       Sets the value of the item specified in the *name* parameter to the short (16-bit) integer value specified in the *value* parameter.

**Throws**      `java.lang.IllegalAccessError`. Occurs when the user tries to set a value to a get-only attribute, or to get a value from a set-only attribute.

`java.lang.NoSuchFieldError`. Occurs when the user tries to set an attribute which is not one of the predefined attributes.

## setString

The `setString` method sets the string value of a specified item.

**Synopsis**    `void setString( String name, String value );`

**Usage**    Sets the value of the item specified in the *name* parameter to the string value specified in the *value* parameter.

**Throws**    `java.lang.IllegalAccessError`. Occurs when the user tries to set a value to a get-only attribute, or to get a value from a set-only attribute.

`java.lang.NoSuchFieldError`. Occurs when the user tries to set an attribute which is not one of the predefined attributes.

## Attributes Names for Get and Set Methods

The attribute names for the `JoltSessionAttributes` class include static final variables (essentially predefined constants) for setting the application address, and for setting and getting timeouts. The predefined attribute names listed in Table 7-2 are used for specifying the String *name* parameter of the `JoltSessionAttributes` get and set methods.

**Note:** The get methods are unrestricted. That is, the programmer can define names in addition to using the predefined ones. The set methods are currently restricted to the use of the predefined names listed in Table 7-2.

**Table 7-2 Predefined Attribute Names**

Attribute Name	Description
APPADDRESS	<p>Defines the location of the machine where the application resides. The format is as follows:</p> <p>String "//host:port"</p> <p>Host is the host machine name or IP address and port is the TCP port number for the Jolt JSL or Jolt Relay. Use <code>setString()</code> to set the value of this attribute.</p>
IDLETIMEOUT	<p>Defines an integer representing the timeout interval in seconds. Set the IDLETIMEOUT to an interval of time that is less than the SESSIONTIMEOUT value.</p> <p>IDLETIMEOUT is used to hint to the system when to drop the network connection while retaining the session. When the IDLETIMEOUT is set to 0, the system is notified that the network connection is always retained. This ensures that when the system times out, the network connection to the target system is terminated, and the session is terminated. The system administrator may override this parameter.</p> <p>Use <code>setIntDef()</code> to set the value of this attribute.</p>
RECVTIMEOUT	<p>Defines the number of seconds to wait after Jolt issues the TCP/IP <code>recv()</code> before timing out. The default is 120 seconds. Use <code>setInt()</code> to set the value of this attribute.</p>
SENDTIMEOUT	<p>Defines the number of seconds to wait after Jolt issues the TCP/IP <code>send()</code> before timing out. The default is 10 seconds.</p>
SESSIONTIMEOUT	<p>SESSIONTIMEOUT is the only attribute used for the <code>get...</code> methods. This attribute is the integer value in minutes of the SESSIONTIMEOUT that is configured in the Jolt server. When a session has been idle for the specified time, the server terminates the session. See the -T option in JSL. Use <code>setString()</code> to set the value of this attribute.</p>

**Note:** For more information about the JSL -T timeout refer to “\*SERVERS Section.”

# JoltSession Class

```
java.lang.Object
|
+----bea.jolt.Session
      |
      +----bea.jolt.JoltSession
```

The `JoltSession` class represents the logon session object and is used to access available TUXEDO services. The GUI-based Jolt Repository handles the propagation of TUXEDO services to Jolt client applications. (For more information about the Jolt Repository, see Chapter 5, “Using the Jolt Repository Editor.”) The programmer must instantiate an object for each logon session. The `JoltSession` object communicates data from, and connects to the TUXEDO System. The session ends with a call using the `endSession()` method.

The `JoltSession` object is passed, by reference, to the `JoltRemoteService` and `JoltTransaction` objects. All Jolt transactions must pass a `JoltSession` to access the TUXEDO application.

```
public class JoltSession

{
    // JoltSession Constructor
    public JoltSession( JoltSessionAttributes attr, String
        userName, String userRole, String userPassword, String
        appPassword )throws SessionException;

    // JoltSession Method
    public void endSession( )throws SessionException;
    public final isAlive( );
    public void onReply ( JoltReply reply );
    protected void finalize( );
}
```

For more information see the following classes: `JoltSessionAttributes`, `JoltRemoteService`, `JoltTransaction`, `JoltReply`.

## JoltSession Constructor

The JoltSession constructor creates a JoltSession object with attributes from the JoltSessionAttributes class. The JoltSession object is the logon to the target system which is identified in the APPADDRESS name of the JoltSessionAttributes. In addition, programmers can specify the IDLETIMEOUT value (in seconds) in JoltSessionAttributes to hint to the system when to drop the network connection while retaining the session. Value 0 means the network connection should be retained throughout the session. The administrator can override the behavior.

The programmer can set all other parameters for this constructor to null, depending upon the authentication level.

## JoltSession

Creates an instance of the JoltSession class with the specified attributes.

**Synopsis**    **JoltSession**( JoltSessionAttributes attr, String userName, String userRole, String userPassword, String appPassword ) throws SessionException;

**Usage**    The JoltSession class creates an instance of the JoltSession class with the specified attributes. Specific components are extracted to assist in joining an application and terminating a session.

JoltSession allows the user identified by the *userName* parameter to log on to the Jolt/TUXEDO system. All parameters must be set according to the authentication level. If the logon is not successful, a SessionException is thrown. The object is returned upon completion.

**Throws**    The JoltSession constructor throws the following exceptions:

SessionException    TPEJOLT: Missing host name or missing port number.  
                          TPEJOLT: Protocol Error  
                          TPEJOLT: Network Error  
                          TPEJOLT: Can't connect to (host name)



The JoltSession constructor throws the following TUXEDO errors:

tpinit(3)	TPEINVAL, TPENOENT, TPEPERM, TPEPROTO, TPESYSTEM, TPEOS
-----------	---

Refer to “TUXEDO Errors” in Appendix A or `tperrno(5)` in the *TUXEDO System Reference Manual* for explanations of these error messages.

## JoltSession Method

The JoltSession class contains one method, the `endSession` method, for handling session logoff activities, two additional methods, `isAlive()` and `onReply()` for handling event subscription, and the overridden `finalize()` method.

### endSession

The `endSession` method performs a session termination procedure.

Synopsis    `void endSession( ) throws SessionException;`

Usage      The `endSession` method is used to terminate the session and obsolete the session object. The `endSession` method allows the user to log off the Jolt/TUXEDO system. When logged off, the session is invalid.

Throws     The `endSession()` method throws the following exceptions:

SessionException	TPEJOLT: Invalid Session TPEJOLT: Connection send error
------------------	--

The `endSession()` method generates a Jolt exception upon receipt of one of the following TUXEDO errors:

tpterm():	TPEPROTO, TPESYSTEM, TPEOS
-----------	----------------------------

See Also    “TUXEDO Errors” in Appendix A or `tperrno(5)` in the *TUXEDO System Reference Manual*.

## isAlive

The **isAlive** method checks if the session is still alive.

**Synopsis**    `boolean isAlive() throws SessionException`

**Usage**    The **isAlive** method will return a boolean (true/false) value indicating whether the client session is valid or not. The validity of the client session is checked at the JoltSessionHandler (JSH). If a client which is operating in connection-less mode calls **isAlive()** while its network connection is closed, this method will cause the network connection to be re-opened and closed.

**Returns**    True if this session is still alive; false otherwise.

**Throws**    The **isAlive()** method throws a `SessionException`.

## onReply

The **onReply()** method is the default event handler for all events.

**Synopsis**    `void onReply( JoltReply reply )`

**Parameter**    *reply*                      Specifies an object containing the data message.

**Usage**    This method is the default event handler for all events. This method is invoked by Jolt and it should be overridden by the user. Client application developers need to provide an implementation of the **onReply()** method to invoke when any notification is received. The `JoltReply` object that is passed to the **onReply()** method contains any data that is included with the notification. The application can use the methods in `JoltReply` and `JoltMessage` to retrieve this data.

Since Jolt is a multi-threaded environment, the **onReply()** method executes on a separate thread simultaneously with other application threads. You must, therefore, write a thread-safe handler. Separate threads could be executing the **onReply()** method concurrently, so the method must be written to accommodate this possibility. Alternately, you can declare the **onReply()** method as synchronized—this will cause all its invocations to be serialized.

**Overrides**    The **onReply()** method overrides an **onReply()** method.

## **finalize**

The `finalize()` method will call the `endSession()` method if an `endSession()` has never been called.

Synopsis    `protected void finalize( )`

Usage      This method is automatically called by the Java garbage collector.

Overrides    The **`finalize()`** method overrides `java.lang.Object.finalize()`.

# JoltRemoteService Class

```

java.lang.Object
|
+----bea.jolt.JoltRequestMessage
      |
      +----bea.jolt.JoltRemoteService

```

The `JoltRemoteService` class is a subclass (child class) of the `JoltRequestMessage` class. It is derived from and inherits the characteristics of its parent class, `JoltRequestMessage`. The `JoltRemoteService` object is reusable; therefore, the programmer should invoke the `JoltRequestMessage` **clear** method to reset any previous data (such as flags, priority, and parameters) before invoking the object again.

`JoltRemoteService` is used to perform the Request/Reply call. To make a call, the programmer:

1. Instantiates an object of this class for each service, using the **add** methods from `JoltRequestMessage` to add the parameters.
2. Invokes the **call** method to send the request.
3. Uses the `JoltRequestMessage` **get** methods, upon successful completion of the call, to retrieve the reply result.

The `JoltRemoteService` methods are described as part of the `JoltRequestMessage` class. Refer to the `JoltRequest Message`, `JoltSession`, and `JoltTransaction` classes for additional information.

```

public class JoltRemoteService extends JoltRequestMessage
{
    public JoltRemoteService( String name, JoltSession s ) throws
        ServiceException;

    // JoltRemoteService methods
    public void call( Transaction t )
        throws ServiceException,
        TransactionException, ApplicationException
}

```

## JoltRemoteService Constructor

The following JoltRemoteService constructor is used to create instances of the JoltRemoteService class.

### JoltRemoteService

The JoltRemoteService constructor defines a constructor for a remote service object.

**Synopsis**     `JoltRemoteService( String name, JoltSession s ) throws  
ServiceException;`

**Usage**     `JoltRemoteService()` defines the constructor for a remote service object. The *name* parameter specifies the service name. The *s* parameter specifies the session object from the JoltSession class. This constructor receives the most **current version** from a repository that holds the service definitions. If the service does not exist, an exception is thrown.

**Throws**     `ServiceExceptions`TPENOENT - Service (service name) is not available.  
  TPEJOLT - Invalid session  
  TPEJOLT - bea.jolt.DefinitionException: Invalid (message)

## JoltRemoteService Methods

The `JoltRemoteService()` method provide the means to call a transaction service for the JoltRemoteService object.

### call

The `call()` method calls the specified service.

**Synopsis**     `void call( Transaction t ) throws ApplicationException,  
TransactionException, ServiceException;`

**Usage** The `call()` method calls the service. The `t` parameter specifies the transaction object. The transaction must belong to the same session as the current service. If a TUXEDO error (not service failure) occurs, the call method throws a `ServiceException`.

You can pass a null to this `call()` method to call services without defining a `JoltTransaction` object or to exclude the services from a transaction.

You can use a `getErrno` to retrieve the TUXEDO error code and error message from the `ServiceException`. If there is a service failure (i.e., `TPESVCFAIL`), the call method throws a runtime exception, `ApplicationException`.

When programming in multithreaded mode, if the transaction is aborted, all outstanding calls associated with such a transaction receive a `TransactionException` with a `TPEABORT` error code. The associated message contains the service name.

To call services without defining a `JoltTransaction` object or to exclude the services from a transaction, you can pass a null to the call method.

**Throws** `ApplicationException` (application code) - “this” object is included.

`ServiceExceptions`

- TPEJOLT: Invalid session
- TPEJOLT: Connection send error
- TPEJOLT: Connection rcv error
- TPEJOLT: Protocol error
- TPEJOLT: bea.jolt.BufException:....
- TPEJOLT: bea.jolt.MessageException:...

`TransactionExceptions`

- TPEJOLT - Invalid Transaction
- TPEABORT - (service name)

The `call()` method generates an exception upon receipt of the following TUXEDO errors:

`tpcall`

- TPEINVAL, TPENOENT, TPEITYPE, TPETRAN,
- TPETIME, TPESVCFAIL, TPESVCERR, TPEBLOCK,
- TPGOTSIG, TPEPROTO, TPESYSTEM, TPEOS

**See Also** “TUXEDO Errors” in Appendix A or `tperrno` in the *TUXEDO System Reference Manual*.

# JoltRequestMessage Abstract Class

JoltRequestMessage is an abstract class whose main purpose is to serve as a base or parent class for [JoltRemoteService](#). The JoltRemoteService class extends from (or is a subclass of) the JoltRequestMessage. All of the methods provided for the JoltRequestMessage class only work with the JoltRemoteService class.

**Note:** All JoltRequestMessage method exceptions are caught by the Java virtual machine by default.

```
public abstract class JoltRequestMessage
{
    public String getName();
    public void setRequestPriority( int priority );
    public int getApplicationCode();
    public void clear();
    public void addByte( String name, byte val );
    public void addShort( String name, short val );
    public void addInt( String name, int val );
    public void addFloat( String name, float val );
    public void addDouble( String name, double val );
    public void addString( String name, String val );
    public void addBytes( String name, byte[] val, int len );
    public void delete( String name );
    public void deleteItem( String name, int itemNo );
    public void setByte( String name, byte value );
    public void setShort( String name, short value );
    public void setInt( String name, int value );
    public void setFloat( String name, float value );
    public void setDouble( String name, double value );
    public void setString( String name, String value );
    public void setBytes( String name, byte[] value, int len );
    public void setByteItem( String name, int itemNo, byte val );
    public void setShortItem( String name, int itemNo, short val );
    public void setIntItem( String name, int itemNo, int val );
    public void setFloatItem( String name, int itemNo, float val );
    public void setDoubleItem( String name, int itemNo, double val );
    public void setStringItem( String name, int itemNo, String val );
    public void setBytesItem( String name, int itemNo, byte[] val,
```

```
int len);
public int  getOccurrenceCount( String name );
public byte getByteDef( String name, byte defValue );
public short getShortDef( String name, short defValue );
public int  getIntDef( String name, int defValue );
public float getFloatDef( String name, float defValue );
public double getDoubleDef( String name, double defValue );
public String getStringDef( String name, String defValue );
public byte[] getBytesDef( String name, byte[] defValue );
public byte getByteItemDef( String name, int itemNo, byte def );
public short getShortItemDef( String name, int itemNo, short
def );
public int  getIntItemDef( String name, int itemNo, int def );
public float getFloatItemDef( String name, int itemNo,
float def );
public double getDoubleItemDef( String name, int itemNo, double
def );
public byte[] getBytesItemsDef( String name, int itemNo, byte[]
def );
public String getStringItemDef( String name, int itemNo, String
def );
}
```



## JoltRequestMessage Methods

The JoltRequestMessage methods are used in conjunction with the abstract JoltRequestMessage class. These methods are divided into five categories: the clear method, get methods, set methods, add methods, and delete methods.

**Note:** The `clear()` method removes JoltRequestMessage settings.

### Get Methods

The get JoltRequestMessage methods are used to get values from the remote service object. These methods encompass two types of get methods, service and request methods and output parameter methods.

**Service and Request Methods.** The service methods handle queries about service attributes such as the name of the current service, or the application code (i.e., `tpusrcode` in ATMI). These methods include:

- ◆ `getApplicationCode()`
- ◆ `getName()`

**Output Parameter Methods.** All of the get methods are used to obtain the value of a named item that must be one of the result parameters. If the item does not exist, the method returns the specified default value. A get method without *itemNo* parameter is equivalent to a get method with *itemNo* 0 (i.e., first occurrence). These methods include the following:

- ◆ `getOccurrenceCount`
- ◆ `getByteDef`
- ◆ `getBytesDef`
- ◆ `getDoubleDef`
- ◆ `getFloatDef`
- ◆ `getIntDef`
- ◆ `getShortDef`
- ◆ `getStringDef`

- ◆ getByteItemDef
- ◆ getBytesItemsDef
- ◆ getDoubleItemDef
- ◆ getFloatItemDef
- ◆ getIntItemDef
- ◆ getShortItemDef
- ◆ getStringItemDef

The corresponding set methods for these get methods are located in the “setByte” section.

## Set Methods

The set methods are used to set values for the remote service object. These methods manipulate the input parameters. The set JoltRequestMessage methods are divided into two categories:

**Service and Request Method.** The service method sets the request priority service attribute, this includes:

- ◆ setRequestPriority

**Input Parameter Methods.** These methods are used to set the value of a named item that must be one of the input parameters. If the named item does not exist, the value is added. These methods include the following:

- ◆ setByte
- ◆ setBytes
- ◆ setDouble
- ◆ setFloat
- ◆ setInt
- ◆ setShort
- ◆ setString

- ◆ `setByteItem`
- ◆ `setBytesItem`
- ◆ `setDoubleItem`
- ◆ `setFloatItem`
- ◆ `setIntItem`
- ◆ `setShortItem`
- ◆ `setStringItem`

For all of the `set...Item` methods, if an item exists, the set method will overwrite it. The parameter *itemNo* specifies an index that points to the value. If there are gaps in the index (that is, you set a value for *itemNo* 3 and there are no values set for *itemNo* 0, 1, and 2) these items will be set with:

- ◆ an empty string, "", for string values
- ◆ 0 for short, int, or byte values
- ◆ 0.0 for floating point and double precision values
- ◆ null for byte arrays

In the following example, the `setStringItem` method is used to change the name John to Jim. The *itemNo* parameter specifies a list of names. Currently, the name John exists for *itemNo* 2. To change the name you might write:

```
setStringItem ("names", 2, "jim");
```

All indexes specified by *itemNo* start at 0 and increment. Therefore, the second item in a list is actually *itemNo* 1 and the first item in the list is actually *itemNo* 0.

## Add and Delete Methods

The add and delete methods are used to add or delete values for the remote service object. These methods manipulate the input parameters. These methods include:

- ◆ `addByte`
- ◆ `addBytes`
- ◆ `addDouble`

- ◆ addFloat
- ◆ addInt
- ◆ addShort
- ◆ addString
- ◆ delete
- ◆ deleteItem

## **clear**

The `clear()` method resets all input/output parameters or any information.

**Synopsis**    `void clear();`

**Usage**      The `clear()` method handles removal of parameter settings or information set by the `JoltRequestMessage` class.

## **getApplicationCode**

The `getApplicationCode()` method gets the application code returned by the service.

**Synopsis**    `int getApplicationCode();`

**Usage**      The `getApplicationCode()` method is equivalent to `tpurcode` or the `rcode` in TUXEDO's `tpreturn(3)`.

**Returns**    This method returns an integer value (32-bit).

## **getName**

The `getName()` method gets the name of the current service.

**Synopsis**    `String getName();`

**Returns**    This method returns a string value.

## getOccurrenceCount

The `getOccurrenceCount()` method gets the number of occurrences of a specified item.

Synopsis    `int getOccurrenceCount( String name );`

Usage      This method retrieves the number of occurrences of an item specified by the *name* parameter. The item must be one of the result or output parameters.

Returns    This method returns an integer value (32-bit).

Throws     `java.lang.NoSuchFieldError`. (field name) Attempt to get a field that is not defined for this service.

## getByteDef

The `getByteDef()` method gets the byte value of a specified parameter.

Synopsis    `byte getByteDef( String name, byte defValue );`

Usage      This method gets the byte value of the *name* parameter or the specified *defValue*, if the name does not exist. The item must be one of the result or output parameters.

Returns    This method returns a byte value (8-bit).

Throws     `java.lang.NoSuchFieldError`. (field name) Attempt to get a field that is not defined for this service.

## getBytesDef

The `getBytesDef()` method gets the byte array value of a specified parameter.

Synopsis    `byte[ ] getBytesDef( String name, byte[] defValue );`

Usage      This method gets the byte array value of the first item specified in the *name* parameter or the specified *defValue*, if the name does not exist.

- Returns** This method returns an array of byte values.
- Throws** `java.lang.NoSuchFieldError`. (field name) Attempt to get a field that is not defined for this service.

## getDoubleDef

The `getDoubleDef()` method gets the double precision value of a specified parameter.

- Synopsis** `double getDoubleDef( String name, double defValue );`
- Usage** This method gets the double precision value of the item specified in the *name* parameter or the specified *defValue*, if the name does not exist.
- Returns** This method returns a double precision value (64-bit).
- Throws** `java.lang.NoSuchFieldError`. (field name) Attempt to get a field that is not defined for this service.

## getFloatDef

The `getFloatDef()` method gets the floating point value of a specified parameter.

- Synopsis** `float getFloatDef( String name, float defValue );`
- Usage** This method gets the floating point value of the item specified in the *name* parameter or the specified *defValue*, if the name does not exist.
- Returns** This method returns a floating point value (32-bit).
- Throws** `java.lang.NoSuchFieldError`. (field name) Attempt to get a field that is not defined for this service.

## getIntDef

The `getIntDef()` method gets the integer value of a specified parameter.

Synopsis    `int getIntDef( String name, int defValue );`

Usage      This method gets the integer value of the item specified in the *name* parameter or the specified *defValue*, if the name does not exist.

Returns    This method returns an integer value (32-bit).

Throws     `java.lang.NoSuchFieldError`. (field name) Attempt to get a field that is not defined for this service.

## getShortDef

The `getShortDef()` method gets the short integer value of a specified parameter.

Synopsis    `short getShortDef( String name, short defValue );`

Usage      This method gets the short integer value (16-bit) of the item specified in the *name* parameter or the specified *defValue*, if the name does not exist.

Returns    This method returns a short integer value (16-bit).

Throws     `java.lang.NoSuchFieldError`. (field name) Attempt to get a field that is not defined for this service.

## getStringDef

The `getStringDef()` method gets the string value of a specified parameter.

Synopsis    `String getStringDef( String name, String defValue );`

Usage      This method gets the string value of the item specified in the *name* parameter or the specified *defValue*, if the name does not exist. The default value can be a null string or any string that you specify.

**Returns** This method returns a string value.

**Throws** `java.lang.NoSuchFieldError`. (field name) Attempt to get a field that is not defined for this service.

## getByteItemDef

The `getByteItemDef()` gets the byte item.

**Synopsis** `byte getByteItemDef( String name, int itemNo, byte def );`

**Usage** This method gets the byte value of the *itemNo* of the *name* parameter. If the item does not exist, the get method returns the default value.

**Returns** This method returns a byte value (8-bit).

**Throws** `java.lang.NoSuchFieldError`. (field name) Attempt to get a field that is not defined for this service.

## getBytesItemsDef

The `getBytesItemDef()` gets the byte array item.

**Synopsis** `byte[] getBytesItemDef( String name, int itemNo, byte[] def );`

**Usage** This method gets the byte value of the *itemNo* of the *name* parameter. If the item does not exist, the get method uses the default value.

**Returns** This method returns a byte value (8-bit).

**Throws** `java.lang.NoSuchFieldError`. (field name) Attempt to get a field that is not defined for this service.



## getDoubleItemDef

The `getDoubleItemDef()` gets the double precision item.

Synopsis     `double getDoubleItemDef( String name, int itemNo, double def );`

Usage        This method gets the double precision value of the *itemNo* of the name parameter. If the item does not exist, the get method returns the default value.

Returns      This method returns a double precision value (64-bit).

Throws       `java.lang.NoSuchFieldError`. (field name) Attempt to get a field that is not defined for this service.

## getFloatItemDef

The `getFloatItemDef()` gets the floating point item.

Synopsis     `float getFloatItemDef( String name, int itemNo, float def );`

Usage        This method gets the floating point value of the *itemNo* of the name parameter. If the item does not exist, the get method returns the default value.

Returns      This method returns a floating point value (32-bit).

Throws       `java.lang.NoSuchFieldError`. (field name) - Attempt to get a field that is not defined for this service.

## getIntItemDef

The `getIntItemDef()` gets the integer item.

Synopsis     `int getIntItemDef( String name, int itemNo, int def );`

Usage        This method gets the integer value of the *itemNo* of the name parameter. If the item does not exist, the get method returns the default value.

- Returns** This method returns an integer value (32-bit).
- Throws** `java.lang.NoSuchFieldError`. (field name) - Attempt to get a field that is not defined for this service.

## getShortItemDef

The `getShortItemDef()` gets the short item.

**Synopsis** `short getShortItemDef( String name, int itemNo, short def );`

**Usage** This method gets the short value of the *itemNo* of the name parameter. If the item does not exist, the get method returns the default value.

**Returns** This method returns a short integer value (16-bit).

**Throws** `java.lang.NoSuchFieldError`. (field name) - Attempt to get a field that is not defined for this service.

## getStringItemDef

The `getStringItemDef()` gets the string value of a specified item.

**Synopsis** `String getStringItemDef( String name, int itemNo, String def );`

**Usage** This method gets the string value of the *itemNo* of the name parameter. If the item does not exist, the get method returns the default value.

**Returns** This method returns a string value.

**Throws** `java.lang.NoSuchFieldError`. (field name) - Attempt to get a field that is not defined for this service.

## setRequestPriority

The `setRequestPriority()` method sets the request priority service attribute.

Synopsis     `void setRequestPriority( int priority );`

Parameter     *priority*                      Specify a priority value between 1 and 100 inclusive.

Usage     Sets the absolute request priority for the current service. It is set until `clear()` is called.

## setByte

The `setByte()` method sets the value of the specified item.

Synopsis     `void setByte( String name, byte value );`

Usage     Sets the value of the item specified in the *name* parameter to the byte value specified in the *value* parameter. The value is an 8-bit byte.

Throws     `java.lang.IllegalAccessError.` (field name) - Attempt to set a value to an output parameter field.

`java.lang.NoSuchFieldException.` (field name) - Attempt to set a field which is not defined for this service.

## setBytes

The `setBytes()` method sets the value of the specified item.

**Synopsis**    `void setBytes( String name, byte[ ] value, int len );`

**Usage**       Sets the value of the item specified in the *name* parameter to the byte array value specified in the *value* parameter, with the length set by the *len* parameter. The value is in 8-bit bytes.

**Throws**      `java.lang.IllegalAccessException.` (field name) - Attempt to set a value to an output parameter field.

`java.lang.NoSuchFieldException.` (field name) - Attempt to set a field which is not defined for this service.

## setDouble

The `setDouble()` method sets the double precision value of the specified item.

**Synopsis**    `void setDouble( String name, double value );`

**Usage**       Sets the value of the item specified in the *name* parameter to the double precision (64-bit) value specified in the *value* parameter.

**Throws**      `java.lang.IllegalAccessException.` (field name) - Attempt to set a value to an output parameter field.

`java.lang.NoSuchFieldException.` (field name) - Attempt to set a field which is not defined for this service.

## setFloat

The `setFloat()` method sets the floating point value of the specified item.

Synopsis    `void setFloat( String name, float value );`

Usage      Sets the value of the item specified in the *name* parameter to the floating point (32-bit) value specified in the *value* parameter.

Throws     `java.lang.IllegalAccessError.` (field name) - Attempt to set a value to an output parameter field.

`java.lang.NoSuchFieldException.` (field name) - Attempt to set a field which is not defined for this service.

## setInt

The `setInt()` method sets the integer value of the specified item.

Synopsis    `void setInt( String name, int value );`

Usage      Sets the value of the item specified in the *name* parameter to the integer (32-bit) value specified in the *value* parameter.

Throws     `java.lang.IllegalAccessError.` (field name) - Attempt to set a value to an output parameter field.

`java.lang.NoSuchFieldException.` (field name) - Attempt to set a field which is not defined for this service.

## setShort

The `setShort()` method sets the short integer value of the specified item.

**Synopsis**    `void setShort( String name, short value );`

**Usage**       Sets the value of the item specified in the *name* parameter to the short (16-bit) integer value specified in the *value* parameter.

**Throws**      `java.lang.IllegalAccessException.` (field name) - Attempt to set a value to an output parameter field.

`java.lang.NoSuchFieldException.` (field name) - Attempt to set a field which is not defined for this service.

## setString

The `setString()` method sets the string value of the specified item.

**Synopsis**    `void setString( String name, String value );`

**Usage**       Sets the value of the item specified in the *name* parameter to the string value specified in the *value* parameter.

**Throws**      `java.lang.IllegalAccessException.` (field name) - Attempt to set a value to an output parameter field.

`java.lang.NoSuchFieldException.` (field name) - Attempt to set a field which is not defined for this service.

## setByteItem

The `setByteItem()` method sets a named item at a specified index with a byte value.

**Synopsis**     `void setByteItem( String name, int itemNo, byte value );`

**Usage**     Sets the item specified in the *name* parameter at the index specified by *itemNo* with the byte value specified by the *value* parameter. If the item already exists, the value is overwritten.

**Throws**     `java.lang.IllegalAccessException.` (field name) - Attempt to set a value to an output parameter field.

`java.lang.NoSuchFieldException.` (field name) - Attempt to set a field which is not defined for this service.

## setBytesItem

The `setBytesItem()` method sets a named item at a specified index with a byte value.

**Synopsis**     `void setBytesItem( String name, int itemNo, byte[] value, int len );`

**Usage**     Sets the item specified in the *name* parameter at the index specified by *itemNo* with the byte array value specified by the *value* parameter. If the item already exists, the value is overwritten.

**Throws**     `java.lang.IllegalAccessException.` (field name) - Attempt to set a value to an output parameter field.

`java.lang.NoSuchFieldException.` (field name) - Attempt to set a field which is not defined for this service.

## setDoubleItem

The `setDoubleItem()` method sets a named item at a specified index with a double precision value.

**Synopsis**    `void setDoubleItem( String name, int itemNo, double value );`

**Usage**       Sets the item specified in the *name* parameter at the index specified by *itemNo* with the double precision value specified by the *value* parameter. If the item already exists, the value is overwritten.

**Throws**      `java.lang.IllegalAccessException.` (field name) - Attempt to set a value to an output parameter field.

`java.lang.NoSuchFieldException.` (field name) - Attempt to set a field which is not defined for this service.

## setFloatItem

The `setFloatItem()` method sets a named item at a specified index with a floating point value.

**Synopsis**    `void setFloatItem( String name, int itemNo, float value);`

**Usage**       Sets the item specified in the *name* parameter at the index specified by *itemNo* with the floating point value specified by the *value* parameter. If the item already exists, the value is overwritten.

**Throws**      `java.lang.IllegalAccessException.` (field name) - Attempt to set a value to an output parameter field.

`java.lang.NoSuchFieldException.` (field name) - Attempt to set a field which is not defined for this service.



## setIntItem

The `setIntItem()` method sets a named item at a specified index with a integer value.

**Synopsis**    `void setIntItem( String name, int itemNo, int value);`

**Usage**    Sets the item specified in the *name* parameter at the index specified by *itemNo* with the integer value specified by the *value* parameter. If the item already exists, the value is overwritten.

**Throws**    `java.lang.IllegalAccessException.` (field name) - Attempt to set a value to an output parameter field.

`java.lang.NoSuchFieldException.` (field name) - Attempt to set a field which is not defined for this service.

## setShortItem

The `setShortItem()` method sets a named item at a specified index with a short integer value.

**Synopsis**    `void setShortItem( String name, int itemNo, short value );`

**Usage**    Sets the item specified in the *name* parameter at the index specified by *itemNo* with the short value specified by the *value* parameter. If the item already exists, the value is overwritten.

**Throws**    `java.lang.IllegalAccessException.` (field name) - Attempt to set a value to an output parameter field.

`java.lang.NoSuchFieldException.` (field name) - Attempt to set a field which is not defined for this service.

## setStringItem

The `setStringItem()` method sets a named item at a specified index with a string value.

**Synopsis**    `void setStringItem( String name, int itemNo, String value );`

**Usage**       Sets the item specified in the *name* parameter at the index specified by *itemNo* with the string value specified by the *value* parameter. If the item already exists, the value is overwritten.

**Throws**      `java.lang.IllegalAccessException.` (field name) - Attempt to set a value to an output parameter field.

`java.lang.NoSuchFieldException.` (field name) - Attempt to set a field which is not defined for this service.

## addByte

The `addByte()` method adds the byte input parameters.

**Synopsis**    `void addByte( String name, byte val );`

**Usage**       The `addByte()` method adds specified byte items to the input parameters.

**Throws**      `java.lang.IllegalAccessException.` (field name) - Attempt to add a value to an input parameter field, or delete a value from an input parameter field.

`java.lang.NoSuchFieldError.` (field name) - Attempt to add or delete a field that is not defined for this service.

## addBytes

The `addBytes()` method adds the byte array input parameter.

Synopsis    `void addBytes( String name, byte[] val, int len );`

Usage      The `addBytes()` method adds the byte value (8-bit) of the item specified in the *name* parameter.

Throws     `java.lang.IllegalAccessException`. (field name) - Attempt to add a value to an input parameter field, or delete a value from an input parameter field.  
  
            `java.lang.NoSuchFieldError`. (field name) - Attempt to add or delete a field that is not defined for this service.

## addDouble

The `addDouble()` method adds the double precision input parameter.

Synopsis    `void addDouble( String name, double val );`

Usage      The `addDouble()` method adds the double precision value (64-bit) of the item specified in the *name* parameter.

Throws     `java.lang.IllegalAccessException`. (field name) - Attempt to add a value to an input parameter field, or delete a value from an input parameter field.  
  
            `java.lang.NoSuchFieldError`. (field name) - Attempt to add or delete a field that is not defined for this service.

## addFloat

The `addFloat()` method adds the floating point input parameter.

**Synopsis**     `void addFloat( String name, float val );`

**Usage**        The `addFloat()` method adds the floating point value (32-bit) of the item specified in the *name* parameter.

**Throws**       `java.lang.IllegalAccessError.` (field name) - Attempt to add a value to an input parameter field, or delete a value from an input parameter field.

`java.lang.NoSuchFieldError.` (field name) - Attempt to add or delete a field that is not defined for this service.

## addInt

The `addInt()` method adds the integer input parameter.

**Synopsis**     `void addInt( String name, int val );`

**Usage**        The `addInt()` method adds the integer value (32-bit) of the item specified in the *name* parameter.

**Throws**       `java.lang.IllegalAccessError.` (field name) - Attempt to add a value to an input parameter field, or delete a value from an input parameter field.

`java.lang.NoSuchFieldError.` (field name) - Attempt to add or delete a field that is not defined for this service.

## addShort

The `addShort()` method adds the short integer value (16-bit) input parameter.

**Synopsis**    `void addShort( String name, short val );`

**Usage**    This method adds the short integer value (16-bit) of the item specified in the *name* parameter.

**Throws**    `java.lang.IllegalAccessError`. (field name) - Attempt to add a value to an input parameter field, or delete a value from an input parameter field.

`java.lang.NoSuchFieldError`. (field name) - Attempt to add or delete a field that is not defined for this service.

## addString

The `addString()` method adds the string value input parameter.

**Synopsis**    `void addString( String name, String val );`

**Usage**    This method adds the string value of the item specified in the *name* parameter.

**Throws**    `java.lang.IllegalAccessError`. (field name) - Attempt to add a value to an input parameter field, or delete a value from an input parameter field.

`java.lang.NoSuchFieldError`. (field name) - Attempt to add or delete a field that is not defined for this service.

## delete

The `delete()` method deletes a first occurrence of the named item or any existing item.

**Synopsis**     `void delete( String name );`

**Usage**        This method deletes the first named item. The named item must be one of the input parameters.

**Throws**       `java.lang.IllegalAccessError.` (field name) - Attempt to add a value to an input parameter field, or delete a value from an input parameter field.

`java.lang.NoSuchFieldError.` (field name) - Attempt to add or delete a field that is not defined for this service.

## deleteItem

The `deleteItem()` method deletes an occurrence of an existing named item.

**Synopsis**     `void deleteItem( String name, int itemNo );`

**Usage**        This method deletes an existing named item (input parameter). If the item does not exist, an exception will be thrown. The `deleteItem` method without the *itemNo* parameter is equivalent to delete methods with *itemNo* 0 (i.e., first occurrence).

**Throws**       `java.lang.IllegalAccessError.` (field name) - Attempt to add a value to an input parameter field, or delete a value from an input parameter field.

`java.lang.NoSuchFieldError.` (field name) - Attempt to add or delete a field that is not defined for this service.

# JoltTransaction Class

```
java.lang.Object
|
+----bea.jolt.Transaction
|
+----bea.jolt.JoltTransaction
```

The JoltTransaction class is the explicit transaction model for Jolt. The JoltTransaction class implements the transaction object. This object can be used by JoltRemoteService to include several services into a single transaction. When a transaction is timed out, the user must rollback the transaction immediately. Due to the current implementation of TUXEDO, only one transaction object can be instantiated at one time.

Refer also to the JoltRemoteService and JoltSession classes.

```
public class JoltTransaction
{
    public JoltTransaction( int timeout, JoltSession s) throws
        TransactionException;
    public void commit() throws TransactionException;
    public void rollback() throws TransactionException;
}
```

# JoltTransaction Constructor

The JoltTransaction Class provides a constructor to create the JoltTransaction object.

## JoltTransaction

The `JoltTransaction()` constructor creates an instance of the JoltTransaction object with the specified parameters.

**Note:** You can pass a null to the `JoltRemoteService.call()` method to call services without defining a JoltTransaction object or to exclude the services from a transaction.

**Synopsis**     `public JoltTransaction( int timeout, JoltSession s) throws  
TransactionException;`

**Usage**     The constructor (or the method that is invoked automatically when a new instance of a class is created) implies the beginning of the transaction. The *s (session)* parameter in the constructor ensures that the transaction does not span over multiple sessions. The current Jolt release allows only one transaction per session.

JoltTransaction requires that you set the timeout for a transaction. Specifying a timeout parameter of 0, sets the timeout to the maximum value for the system. If the transaction is not completed within this period of time (the time between the `tpbegin()` and the `tpcommit()`), then Jolt generates a `TransactionException`.

The `RECVTIMEOUT` for each transactional `JoltRemoteService.call()` is automatically adjusted to the proper timeout value.

**Throws**     `TransactionException`    `TPEJOLT: Invalid session`  
                                      `TPEJOLT: Connection send error`  
                                      `TPEJOLT: Connection recv error`  
                                      `TPEJOLT: Protocol error`

The JoltTransaction constructor generates an exception upon receipt of the following TUXEDO errors:

`tpbegin`                            `TPEINVAL, TPETRAN, TPEPROTO, TPESYSTEM, TPEOS`



See Also “TUXEDO Errors” in Appendix A or `tperrno(5)` in the *TUXEDO System Reference Manual*.

## JoltTransaction Methods

The `JoltTransaction` methods provide the means to start or end a transaction process. The following methods handle commit and rollback transaction processing for the `JoltTransaction` class.

### **commit**

The `commit()` method performs the transaction commit.

Synopsis    `void    commit()`

Usage      After `commit()` is called, the object is obsolete.

Throws     `TransactionException`    `TPEJOLT`: Invalid transaction  
    `TPEJOLT`: Connection send error  
    `TPEJOLT`: Connection recv error  
    `TPEJOLT`: Protocol error  
    `TPEABORT`: Requests pending

The `commit()` method generates an exception upon receipt of the following TUXEDO errors:

<code>tpcommit(3)</code>	<code>TPEINVAL</code> , <code>TPETIME</code> , <code>TPEABORT</code> , <code>TPEHEURISTIC</code> , <code>TPEHAZARD</code> , <code>TPEPROTO</code> , <code>TPESYSTEM</code> , <code>TPEOS</code>
--------------------------	--

## rollback

The `rollback()` method aborts the transaction.

**Synopsis**    `void    rollback()`

**Usage**      After `rollback()` is called, the object is obsolete.

**Throws**    `TransactionException`    TPEJOLT: Invalid transaction  
   TPEJOLT: Connection send error  
   TPEJOLT: Connection recv error  
   TPEJOLT: Protocol error

The `rollback()` method generates an exception upon receipt of the following TUXEDO errors:

`tpabort(3)`                    TPEINVAL, TPEHEURISTIC, TPEHAZARD, TPEOS,  
   TPEPROTO, TPESYSTEM

**See Also**    “TUXEDO Errors” in Appendix A or `tperrno(5)` in the *TUXEDO System Reference Manual*.

# JoltEvent Class

```
java.lang.Object
|
+----bea.jolt.JoltEvent
```

The `JoltEvent` class extends the `java.lang.Object` class and is a base class for various event subscriptions. This class is not designed to be instantiated directly. The `JoltEvent` class provides some common implementations for all subscriptions. An event can be a notification event or service event. A notifiable event generates async notification (unsolicited notification or event notification) while a service event invokes a service.

For additional information, refer also to the `JoltSession` class.

```
public class JoltEvent

    unsubscribe()
    unsubscribeAll(Session)
```

## JoltEvent Methods

The following methods are used with the `JoltEvent` class.

### unsubscribe

Deletes the subscription to an event.

**Synopsis**    `public int unsubscribe()`

**Usage**    This method is used to stop subscribing to the event specified in the constructor. Once it is unsubscribed, this object becomes obsoleted. All notifications received as a result of a subscription will cause the `onReply()` method in the session to be invoked.

- Returns**     Number of subscriptions deleted.
- Throws**     `EventException`. No such event or invalid event.
- `SessionException`. An error occurs in this session.

## **unsubscribeAll**

The `unsubscribeAll()` method unsubscribes all event subscriptions in the specified session.

**Synopsis**     `public static int unsubscribeAll(Session session)`

**Usage**       Unsubscribe all event subscriptions in the specified session. The *session* parameter requires a Jolt session object.

**Returns**     Number of subscriptions deleted.

**Throws**     `EventException`. Unsubscription error from TUXEDO.

`SessionException`. Invalid session or a session error.

# JoltUserEvent Class

```
java.lang.Object
|
+----bea.jolt.JoltEvent
      |
      +----bea.jolt.JoltUserEvent
```

The `JoltUserEvent` class extends `JoltEvent`. `JoltUserEvent` implements a subscription to an asynchronous (async) notification event. An async notification is either an unsolicited event notification or event notification from the TUXEDO Event Broker. Unsolicited notifications are produced in response to a TUXEDO `tpnotify()` call or a TUXEDO `tpbroadcast()` call. Event notifications are produced as a result of a TUXEDO `tppost()` call.

The `JoltUserEvent` class is used in Jolt 1.1 to support notification. This class provides support for both unsolicited notification (produced as a result of `tpnotify()` or `tpbroadcast()`) and event notifications (produced as a result of `tppost()`).

The String `JoltUserEvent.UNSOLMSG` in the class is a constant which the application programmer uses to request unsolicited messages.

**Note:** In TUXEDO, an unsolicited notification is indistinguishable from event notification. This is also reflected in Jolt.

The handler for unsolicited notification and event notification is done in the `Session` object. A thread is used in this class to monitor any incoming messages.

For additional information, refer also to the `JoltSession`, `JoltMessage`, and `JoltReply` classes.

```
public class JoltUserEvent
    UNSOLMSG
    JoltUserEvent(String, String, Session)
```

## UNSOLMSG

The regular expression constant for unsolicited notification subscription.

Synopsis    `final static String UNSOLMSG`

Parameters	expr	Notes
<code>event</code>	<code>JoltUserEvent.UNSOLMSG</code> for unsolicited notification or a	

*filter* null or boolean expression. The parameter *filter* is a String of the

*session* a JoltSession object. The parameter session is the JoltSession to

**Usage** This constructor subscribes the specific asynchronous notification. An asynchronous

This constructor will return a `JoinUserEvent` object which will provide notification

**Throws**     `SessionException`. Invalid session or a session error.

`EventException`. Filter for unsolicited subscription is not null, or event subscription failed.

# JoltReply Class

```
java.lang.Object
|
+----bea.jolt.JoltReply
```

The JoltReply class extends the `java.lang.Object`. JoltReply is a place holder of the message for unsolicited messages or event notifications. This class provides the application with access to any message received with a TUXEDO event or notification.

```
public class JoltReply
    Message getMessage()
```

## JoltReply Methods

The following method is used with the JoltReply class.

### getMessage

Gets the response message.

**Synopsis**     Message **getMessage()**

**Usage**     The `getMessage()` method returns a JoltMessage object. The returned object provides the application with access to any data that is associated with the event or notification.

**Returns**     The `getMessage()` method returns a message object.



# JoltMessage Class

```
java.lang.Object
|
+----bea.jolt.Message
      |
      +----bea.jolt.JoltMessage
```

The `JoltMessage` class extends `java.lang.Object`. This class implements the `Message` class, which encapsulates the attribute-value pair data for the application protocol. This class allows the user to get an output attribute to the message. For more information, refer also to the `JoltReply` classes.

```
public class JoltMessage

//These methods are duplicates of the JoltRequestMessage class
//They are inherited from that class.

    public int getOccurrenceCount(String name);
    public byte getByteDef(String name, byte def);
    public short getShortDef(String name, short def);
    public int getIntDef(String name, int def);
    public float getFloatDef(String name, float def);
    public double getDoubleDef(String name, double def);
    public String getStringDef(String name,String def);
    public byte[] getBytesDef(String name, byte def[]);
    public byte getByteItemDef(String name, int itemNo, byte def);
    public short getShortItemDef(String name, int itemNo, short def);
    public int getIntItemDef(String name, int itemNo, int def);
    public float getFloatItemDef(String name, int itemNo, float def);
    public double getDoubleItemDef(String name, int itemNo, double
def);
    public byte[] getBytesItemDef(String name, int itemNo, byte
def[]);
    public String getStringItemDef(String name, int itemNo, String
def);
```

## JoltMessage Methods

The following methods are used with the JoltMessage class.

### getOccurrenceCount

Get the number of occurrence of a named item.

**Synopsis**     `synchronized int getOccurrenceCount(String name)`

**Parameters**     *name*                      The name of the item.

**Throws**     `NoSuchFieldError`. It is an invalid name.

### getBytesDef

Get the first item based on its name.

**Synopsis**     `public byte getBytesDef(String name, byte def)`

**Parameters**     *name*                      The name of the item.

*def*                      The default byte value.

**Usage**     Get the first item based on its name. If it does not exist, the default value will be returned.

**Throws**     `NoSuchFieldError`. It is an invalid name.

## getShortDef

Get the first item based on its name.

Synopsis     `short getShortDef(String name, short def)`

Parameters	<i>name</i>	The name of the item.
	<i>def</i>	The default short value.

Usage     Get the first item based on its name. If it does not exist, the default value will be returned.

Throws     `NoSuchFieldError`. It is an invalid name.  
              `IllegalAccessError`. Cannot delete an input item.

## getIntDef

Get the first item based on its name.

Synopsis     `int getIntDef(String name, int def)`

Parameters	<i>name</i>	The name of the item.
	<i>def</i>	The default int value.

Usage     Get the first item based on its name. If it does not exist, the default value will be returned.

Throws     `NoSuchFieldError`. It is an invalid name.  
              `IllegalAccessError`. Cannot delete an input item.

## getFloatDef

Get the first item based on its name.

**Synopsis**     `float getFloatDef(String name, float def)`

<b>Parameters</b>	<i>name</i>	The name of the item.
	<i>def</i>	The default float value.

**Usage**     Get the first item based on its name. If it does not exist, the default value will be returned.

**Throws**     `NoSuchFieldError`. It is an invalid name.

## getDoubleDef

Get the first item based on its specified name.

**Synopsis**     `double getDoubleDef(String name, double def)`

<b>Parameters</b>	<i>name</i>	The name of the item.
	<i>def</i>	The default double value.

**Usage**     Get the first item based on its specified name. If it does not exist, the default value is returned.

**Throws**     `NoSuchFieldError`. It is an invalid name.

## getStringDef

Get the first item based on its name.

**Synopsis**     `String getStringDef(String name,String def)`

<b>Parameters</b>	<i>name</i>	The name of the item.
	<i>def</i>	The default string value.

Usage Get the first item based on its name. If it does not exist, the default value will be returned.

Throws `NoSuchFieldError`. It is an invalid name.

## getBytesDef

Get the first item based on its name.

Synopsis `byte[] getBytesDef(String name, byte def[])`

Parameters *name* The name of the item.

*def* The default byte-array value.

Usage Get the first item based on its name. If it does not exist, the default value will be returned.

Throws `NoSuchFieldError`. It is an invalid name.

## getBytesItemDef

Get an occurrence of a named item of byte data type.

Synopsis `byte getBytesItemDef(String name, int itemNo, byte def)`

Parameters *name* Name of the item.

*itemNo* Occurrence number of the item.

*def* Default value.

Usage The occurrence starts from 0. If it doesn't exist, the default value will be returned.

Returns A byte value.

Throws `NoSuchFieldError`. It is an invalid name.

## getShortItemDef

Get an occurrence of a named item of short data type.

**Synopsis**     `short getShortItemDef(String name, int itemNo, short def)`

<b>Parameters</b>	<i>name</i>	Name of the item.
	<i>itemNo</i>	Occurrence number of the item.
	<i>def</i>	Default value.

**Usage**     The occurrence starts from 0. If it doesn't exist, the default value will be returned.

**Returns**     A short value.

**Throws**     `NoSuchFieldError`. It is an invalid name.

## getIntItemDef

Get an occurrence of a named item of int data type.

**Synopsis**     `int getIntItemDef(String name, int itemNo, int def)`

<b>Parameters</b>	<i>name</i>	Name of the item.
	<i>itemNo</i>	Occurrence number of the item.
	<i>def</i>	Default value.

**Usage**     The occurrence starts from 0. If it doesn't exist, the default value will be returned.

**Returns**     An integer value.

**Throws**     `NoSuchFieldError`. It is an invalid name.

## getFloatItemDef

Get an occurrence of a named item of float data type.

Synopsis     `float getFloatItemDef(String name, int itemNo, float def)`

Parameters	<i>name</i>	Name of the item.
	<i>itemNo</i>	Occurrence number of the item.
	<i>def</i>	Default value.

Usage     The occurrence starts from 0. If it doesn't exist, the default value will be returned.

Returns    A floating point value.

Throws     `NoSuchFieldError`. It is an invalid name.

## getDoubleItemDef

Get an occurrence of a named item of double data type.

Synopsis     `double getDoubleItemDef(String name, int itemNo, double def)`

Parameters	<i>name</i>	Name of the item.
	<i>itemNo</i>	Occurrence number of the item.
	<i>def</i>	Default value.

Usage     The occurrence starts from 0. If it doesn't exist, the default value will be returned.

Returns    A double precision value.

Throws     `NoSuchFieldError`. It is an invalid name.

## getBytesItemDef

Get an occurrence of a named item of byte-array data type.

**Synopsis**     `byte[] getBytesItemDef(String name, int itemNo, byte def[])`

<b>Parameters</b>	<i>name</i>	Name of the item.
	<i>itemNo</i>	Occurrence number of the item.
	<i>def</i>	Default value.

**Usage**     The occurrence starts from 0. If it doesn't exist, the default value will be returned.

**Returns**     A byte-array object.

**Throws**     `NoSuchFieldError`. It is an invalid name.

## getStringItemDef

Get an occurrence of a named item of string data type.

**Synopsis**     `String getStringItemDef(String name, int itemNo, String def)`

<b>Parameters</b>	<i>name</i>	Name of the item.
	<i>itemNo</i>	Occurrence number of the item.
	<i>def</i>	Default value.

**Usage**     The occurrence starts from 0. If it doesn't exist, the default value will be returned.

**Returns**     A string value.

**Throws**     `NoSuchFieldError`. It is an invalid name.







# A Jolt Class Library Errors and Exceptions

This appendix describes the Jolt Class Library errors and exceptions. The Jolt Class Library returns both Jolt and TUXEDO errors and exceptions. The Jolt Class Library errors and exceptions are also listed for each class, constructor, and method listed in Chapter 7, “Jolt Class Library Reference.”

The following topics are included in this appendix:

- ◆ Jolt Error and Exception Handling
- ◆ ApplicationException Class
- ◆ JoltException Class
- ◆ EventException Class
- ◆ MessageException Class
- ◆ ServiceException Class
- ◆ SessionException Class
- ◆ TransactionException Class
- ◆ TUXEDO Errors

TUXEDO errors are described briefly in this appendix. For a complete explanation of TUXEDO errors, refer to the *TUXEDO System Reference Manual*.

# Jolt Error and Exception Handling

An error condition indicates that a non-recoverable error has occurred. When an error occurs, a message is displayed and the current method stops executing.

**Note:** In general, these errors are not caught in your application. Normally, Java errors and exceptions are automatically caught by Jolt.

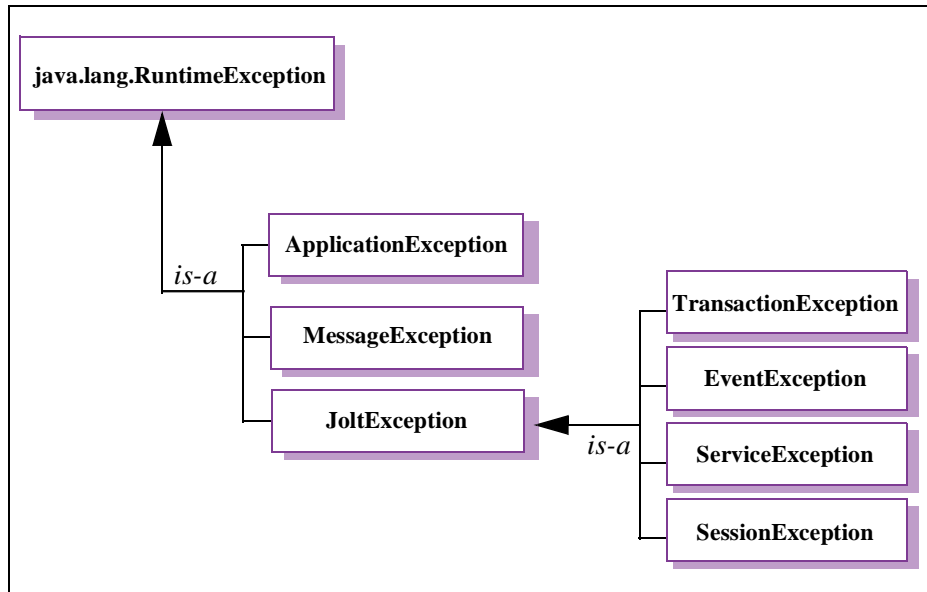
An exception indicates that a condition occurred that requires special handling to prevent the application from terminating. Exceptions can be caught and handled.

Each exception or error comes with an error code and a simple text message. The following table describes the relationship between the error code types and their messages.

**Table A-1 Error Code Text Messages**

If the error code	Then the text message
Is one of the TUXEDO errors (e.g., TPENOENT)	Is the text message from ATMI <code>tpsterror(3)</code> .
TPEJOLT	Provides useful information about what error has occurred and location. You can use a <code>getErrno</code> to get the error number and a <code>getMessage</code> to obtain the error message for the exception object.

The Jolt Class Library uses exceptions to report runtime problems. All Jolt exceptions extend from Java's `RuntimeException` and `Error` classes. Figure A-1 illustrates the relationship of the `JoltException` classes to the Java `RuntimeException` class.

**Figure A-1 Jolt Exception Class Hierarchy**

Since Java's `RuntimeException` and `Error` conditions are caught automatically by the Java virtual machine (VM), programmers can ignore these and rely on the Java VM to trap them. However, programmers must catch the exceptions or errors that are important to their particular applications so that application-specific recovery can be initiated.

# ApplicationException Class

```
java.lang.Object
|
+----java.lang.Throwable
      |
      +----java.lang.Exception
            |
            +----java.lang.RuntimeException
                  |
                  +----bea.jolt.ApplicationException
```

The ApplicationException class is used by the JoltRemoteService class and its methods. The ApplicationException is thrown only when the service calls a tpreturn (TPFAIL, tpurcode, ...).

```
public class ApplicationException extends
java.lang.RuntimeException
```

## ApplicationException Methods

The following methods are used with the ApplicationException.

### getMessage Method

The getMessage method gets the error message.

**Synopsis**    `public String getMessage();`

**Usage**    This method is taken from the Java RuntimeException class.

### getApplicationCode Method

The getApplicationCode method gets the application code (tpurcode).

**Synopsis**    `public int getApplicationCode();`

### getObject Method

The getObject method gets the object that throws this exception.

**Synopsis**    `public Object getObject();`

**Usage**    In Jolt 1.1, the object retrieved by getObject is an instance of the JoltRemoteService class. The caller can still use this object to retrieve all the output parameters. See getStringDef() or similar in JoltRemoteService class.

# JoltException Class

```
java.lang.Object
|
+----java.lang.Throwable
|
+----java.lang.Exception
|
+----java.lang.RuntimeException
|
+----bea.jolt.JoltException
```

All Jolt exceptions are derived from the Java RuntimeException class. The Java virtual machine catches them by default, thus the programmer is not required to catch any of these exceptions.

```
public class JoltException extends java.lang.RuntimeException
{
    public final static int TPEABORT = 1; // Tuxedo error codes
    public final static int TPEBADDESC = 2;
    public final static int TPEBLOCK = 3;
    public final static int TPEINVAL = 4;
    public final static int TPELIMIT = 5;
    public final static int TPENOENT = 6;
    public final static int TPEOS = 7;
    public final static int TPEPERM = 8;
    public final static int TPEPROTO = 9;
    public final static int TPESVCERR = 10;
    public final static int TPESVCFail = 11;
    public final static int TPESYSTEM = 12;
    public final static int TPETIME = 13;
    public final static int TPETRAN = 14;
    public final static int TPGOTSIG = 15;
    public final static int TPERMERR = 16;
    public final static int TPEITYPE = 17;
    public final static int TPEOTYPE = 18;
    public final static int TPERELEASE = 19;
    public final static int TPEHAZARD = 20;
    public final static int TPEHEURISTIC = 21;
```

```
public final static int TPEEVENT = 22;
public final static int TPEMATCH = 23;
public final static int TPEDIAGNOSTIC = 24;
public final static int TPEMIB = 25;
public final static int TPEJOLT = 100; // Jolt error or
                                     /programming error
```

## JoltException Methods

The JoltException methods are used to obtain information about the error or the condition that causes the error.

### getMessage Method

The getMessage method gets the error message.

Synopsis    `public String getMessage();`

Usage      This method is taken from the Java RuntimeException class.

### getErrno Method

The getErrno method gets the error code.

Synopsis    `public int getErrno();`

### getObject Method

The getObject method gets the object which throws this exception.

Synopsis    `public Object getObject();`

Usage      In some situations, this method returns a “null” object.



## EventException Class

```
java.lang.Object
|
+----java.lang.Throwable
      |
      +----java.lang.Exception
            |
            +----java.lang.RuntimeException
                  |
                  +----bea.jolt.JoltException
                        |
                        +----bea.jolt.EventException
```

This exception is thrown when the user encounters any error during the event subscription.

```
public class EventException extends JoltException
```

## MessageException Class

```
java.lang.Object
|
+----java.lang.Throwable
      |
      +----java.lang.Exception
            |
            +----java.lang.RuntimeException
                  |
                  +----bea.jolt.MessageException
```

This exception is thrown when there is an error in parsing the internal message object.

```
public class MessageException
extends RuntimeException
```

# ServiceException Class

```
java.lang.Object
|
+----java.lang.Throwable
      |
      +----java.lang.Exception
            |
            +----java.lang.RuntimeException
                  |
                  +----bea.jolt.JoltException
                        |
                        +----bea.jolt.ServiceException
```

The ServiceException class extends JoltException. The ServiceException class is used by the JoltRemoteService class and its methods.

```
public class ServiceException extends JoltException
```

# SessionException Class

```
java.lang.Object
|
+----java.lang.Throwable
      |
      +----java.lang.Exception
            |
            +----java.lang.RuntimeException
                  |
                  +----bea.jolt.JoltException
                        |
                        +----bea.jolt.SessionException
```

The SessionException extends JoltException. The SessionException is used by the JoltSession class and its methods.

```
public class SessionException extends JoltException
```

## TransactionException Class

```
java.lang.Object
|
+----java.lang.Throwable
      |
      +----java.lang.Exception
            |
            +----java.lang.RuntimeException
                  |
                  +----bea.jolt.JoltException
                        |
                        +----bea.jolt.TransactionException
```

The TransactionException class extends JoltException. The TransactionException class is used by the JoltTransaction class and class methods.

```
public class TransactionException extends JoltException
```

# TUXEDO Errors

Expanded references to TUXEDO will be available in a future release of the Jolt product documentation. If you require an immediate, expanded reference for TUXEDO related errors, see the *BEA TUXEDO Reference Manual*.

**Table A-2 TUXEDO Errors**

Error	Description
TPEABORT	A transaction could not commit because the work performed by the initiator, or by one or more of its participants, could not commit.
TPEBADDESC	A call descriptor is invalid or is not the descriptor with which a conversational service was invoked.
TPEBLOCK	A blocking condition exists and TPNOBLOCK was specified.
TPEDIAGNOSTIC	To be determined.
TPEEVENT	An event occurred; the event type is returned in revent.
TPEHAZARD	Due to a failure, the work done on behalf of the transaction can have been heuristically completed.
TPEHEURISTIC	Due to a heuristic decision, the work done on behalf of the transaction was partially committed and partially aborted.
TPEINVAL	An invalid argument was detected.
TPEITYPE	The type and subtype of the input buffer is not one of the types and subtypes that the service accepts.
TPELIMIT	The caller's request was not sent because the maximum number of outstanding requests or connections has been reached.
TPEMATCH	svcname is already advertised for the server but with a function other than func.
TPEMIB	To be determined.
TPENOENT	Cannot send to svc because it does not exist or is not the correct type of service.

**Table A-2 TUXEDO Errors**

<b>Error</b>	<b>Description</b>
TPEOS	An operating system error has occurred.
TPEOTYPE	The type and subtype of the reply are not known to the caller.
TPEPERM	A client cannot join an application because it does not have permission to do so or because it has not supplied the correct application password.
TPEPROTO	A library routine was called in an improper context.
TPERELEASE	To be determined.
TPERMERR	A resource manager failed to open or close correctly.
TPESVCERR	A service routine encountered an error either in tpreturn(3) or tpforward(3). For example, bad arguments were passed.
TPESVCFAIL	The service routine sending the caller's reply called.
TPESYSTEM	A System/T error occurred.
TPETIME	A time-out occurred.
TPETRAN	The caller cannot be placed in transaction mode.
TPGOTSIG	A signal was received and TPSIGRSTRT was not specified.



# B System Messages

Jolt system messages and code references will be available in a future release of the Jolt product documentation. If you require an immediate, expanded reference, refer to the *TUXEDO System Message Manual, Volume 2*.

This appendix includes:

- ◆ Jolt System Messages
- ◆ Repository Messages
- ◆ FML Error Messages
- ◆ Information Messages
- ◆ Jolt Relay Adapter (JRAD) Messages
- ◆ Jolt Relay (JRLY) Messages
- ◆ Bulk Loader Utility Messages

# Jolt System Messages

<b>1503 ERROR</b>	<b>Could not initialize Jolt administration services.</b>	
	<b>Description</b>	Jolt administration services cannot be started.
	<b>Action</b>	Check the userlog for other messages to determine the proper course of action.
	<b>See Also</b>	<i>TUXEDO Administration Guide</i>
<b>1504 ERROR</b>	<b>Failed to advertise local Jolt administration service &lt;service name&gt;.</b>	
	<b>Description</b>	Jolt administration services cannot be started.
	<b>Action</b>	Check the userlog for other messages to determine the proper course of action.
	<b>See Also</b>	<i>TUXEDO Administration Guide</i>
<b>1505 ERROR</b>	<b>Failed to advertise global Jolt administration service &lt;service name&gt;.</b>	
	<b>Description</b>	Jolt administration services cannot be started.
	<b>Action</b>	Check the userlog for other messages to determine the proper course of action.
	<b>See Also</b>	<i>TUXEDO Administration Guide</i>
<b>1506 ERROR</b>	<b>Terminating Jolt administration services in preparation for shutdown.</b>	
	<b>Description</b>	The JSL has completed its shutdown and is exiting the system.
	<b>Action</b>	Informational message, no action required.
	<b>See Also</b>	<i>TUXEDO Administration Guide</i>

<b>1510 ERROR</b>	<b>Received network message with unknown context.</b>	
	<b>Description</b>	BEA Jolt protocol failure. Received a corrupted or an improper message.
	<b>Action</b>	Restart Jolt client.
<b>1511 ERROR</b>	<b>_tprandkey() failed tperno = %d, could not generate random encryption key.</b>	
	<b>Description</b>	TUXEDO internal failure.
	<b>Action</b>	Restart Jolt servers.
<b>1512 ERROR</b>	<b>Sending of reply to challenge call to client failed.</b>	
	<b>Description</b>	JSH was unable to reply to Jolt client due to network error.
	<b>Action</b>	Restart client.
<b>1513 ERROR</b>	<b>Failed to encrypt ticket information.</b>	
	<b>Description</b>	BEA TUXEDO internal failure.
	<b>Action</b>	Retry the option. If the problem persists, contact BEA Technical Support.
<b>1514 ERROR</b>	<b>Incorrect ticket value sent by workstation client.</b>	
	<b>Description</b>	BEA Jolt protocol failure.
	<b>Action</b>	Retry the option. If the problem persists, contact BEA Technical Support.
<b>1515 ERROR</b>	<b>Tried to process unexpected message opcode 0x%1x.</b>	
	<b>Description</b>	BEA Jolt protocol failure. Client is sending Jolt messages with unknown opcodes.
	<b>Action</b>	Retry the option. If the problem persists, contact BEA Technical Support.
<b>1516 ERROR</b>	<b>Unrecognized message format, release %1d.</b>	
	<b>Description</b>	BEA Jolt protocol failure.
	<b>Action</b>	Make sure the client classes are at the appropriate version level.



<b>1517 ERROR</b>	<b>Commit handle and clientid have no matching requests.</b>	
	<b>Description</b>	Received a copy from TUXEDO that has no corresponding client.
	<b>Action</b>	No action required.
<b>1518 ERROR</b>	<b>Call handle and clientid have no matching requests.</b>	
	<b>Description</b>	Received a reply from TUXEDO that has no corresponding client.
	<b>Action</b>	No action required.
<b>1519 ERROR</b>	<b>Application password does not match.</b>	
	<b>Description</b>	Authentication error.
	<b>Action</b>	Check the application password.
<b>1521 ERROR</b>	<b>Unrecognized message magic %ld.</b>	
	<b>Description</b>	Inappropriate message is sent to JSH/JSL.
	<b>Action</b>	Check the client sending erroneous messages.
<b>1522 ERROR</b>	<b>Memory allocation failure.</b>	
	<b>Description</b>	Machine does not have enough memory.
	<b>Action</b>	Check the machine resources.
<b>1523 ERROR</b>	<b>Memory allocation failure.</b>	
	<b>Description</b>	Machine does not have enough memory.
	<b>Action</b>	Check the machine resources.
<b>1524 ERROR</b>	<b>Failed to create encryption/decryption schedule.</b>	
	<b>Description</b>	BEA TUXEDO internal error.
	<b>Action</b>	Retry the option. If the problem persists, contact BEA Technical Support.

<b>1525 ERROR</b>	<b>Tried to process unexpected message opcode 0x%1x.</b>	
	<b>Description</b>	Received a message with invalid opcode.
	<b>Action</b>	Check the client.
<b>1526 ERROR</b>	<b>Jolt license has expired.</b>	
	<b>Description</b>	License for Jolt use has expired.
	<b>Action</b>	Contact BEA Technical Support.
<b>1527 ERROR</b>	<b>Expected argument to -c option.</b>	
	<b>Description</b>	Option -c needs an argument.
	<b>Action</b>	Provide a valid argument.
<b>1528 ERROR</b>	<b>Request for inappropriate session type.</b>	
	<b>Description</b>	Received a message without valid session information.
	<b>Action</b>	Restart the client.
<b>1529 ERROR</b>	<b>Session type must be <b>RETAINED</b> or <b>TRANSIENT</b>.</b>	
	<b>Description</b>	Server configuration does not match client request.
	<b>Action</b>	Check the -c argument of the JSL.
<b>1530 ERROR</b>	<b>Received RECONNECT message with invalid context.</b>	
	<b>Description</b>	Client context is cleaned. A -T option is specified to the JSL.
	<b>Action</b>	Check the -T option. Check the network errors also.
<b>1531 ERROR</b>	<b>Received invalid RECONNECT request</b>	
	<b>Description</b>	Received a RECONNECT request.
	<b>Action</b>	Restart client.

<b>1532 ERROR</b>	<b>Received J_CLOSE message with invalid context.</b>	
	<b>Description</b>	Timeout in connection.
	<b>Action</b>	If a request is sent after a timeout that is longer than the session timeout of the JSL, the JSH cannot validate the session ID.
<b>1533 ERROR</b>	<b>Sending of reply of close protocol failed.</b>	
	<b>Description</b>	BEA Jolt protocol failure.
	<b>Action</b>	Check the client.
<b>1534 ERROR</b>	<b>Sending of reply of reconnect protocol failed.</b>	
	<b>Description</b>	BEA Jolt protocol failed.
	<b>Action</b>	Check the client.
<b>1535 ERROR</b>	<b>Timestamp mismatch in close protocol.</b>	
	<b>Description</b>	BEA Jolt protocol failed.
	<b>Action</b>	Restart the client.
<b>1536 ERROR</b>	<b>Received J_RECONNECT message with invalid context.</b>	
	<b>Description</b>	BEA Jolt protocol failed. Session timed out before RECONNECT request arrived.
	<b>Action</b>	Restart the client.
<b>1537 ERROR</b>	<b>Timestamp mismatch in reconnect protocol.</b>	
	<b>Description</b>	BEA Jolt protocol failure.
	<b>Action</b>	Restart the client.
<b>1538 ERROR</b>	<b>Client address mismatch in reconnect protocol.</b>	
	<b>Description</b>	BEA Jolt protocol failure.
	<b>Action</b>	Restart the client.

<b>1539 ERROR</b>	<b>Failed to decrypt reconnect information.</b>	
	<b>Description</b>	BEA Jolt protocol failure.
	<b>Action</b>	Restart the client.
<b>1540 ERROR</b>	<b>Failed to encrypt reconnect information.</b>	
	<b>Description</b>	BEA Jolt protocol failure.
	<b>Action</b>	Restart the client.
<b>1541 ERROR</b>	<b>Received RECONNECT request for nonTRANSIENT client.</b>	
	<b>Description</b>	Improper request from client.
	<b>Action</b>	Restart the client.
<b>1542 ERROR</b>	<b>Unlicensed Jolt server.</b>	
	<b>Description</b>	The JSL is not licensed. The installation is incomplete, or it failed to burn the license into the JSL.
	<b>Action</b>	Reinstall Jolt with a valid Jolt license.
<b>1543 ERROR</b>	<b>Invalid Jolt license.</b>	
	<b>Description</b>	The license used for the Jolt installation is not for the Jolt product. The TUXEDO license may have been used during installation instead of the Jolt license.
	<b>Action</b>	Reinstall Jolt with a valid Jolt license.
<b>1544 ERROR</b>	<b>This TUXEDO is not Release &lt;TUXEDO release number&gt;.</b>	
	<b>Description</b>	Jolt is compatible with TUXEDO Release 6.1 or 6.2. The JSL has determined that the TUXEDO release is not compatible.
	<b>Action</b>	Install TUXEDO 6.1 or TUXEDO 6.2.

<b>1545 ERROR</b>	<b>Cannot determine if this TUXEDO is &lt;TUXEDO release number&gt;: service.TMIB failed.</b>	
	<b>Description</b>	This version of TUXEDO does not support the MIB. The TUXEDO release may be TUXEDO 6.0 or earlier.
	<b>Action</b>	Install TUXEDO 6.1 or 6.2 or check to ensure that your TUXEDO release is 6.1 or 6.2.
<b>1546 WARN</b>	<b>The version of this TUXEDO is not available; &lt;TUXEDO release number&gt; is assumed.</b>	
	<b>Description</b>	The MIB is supported with this version of TUXEDO, but the release number is unavailable. The TUXEDO version might not be a master binary. It might also be an internal version of TUXEDO.
	<b>Action</b>	No action is required.
<b>1547 ERROR</b>	<b>Memory allocation failure in JOLT_SUBSCRIBE.</b>	
	<b>Description</b>	Check resources of the machine.
	<b>Action</b>	Restart TUXEDO after increasing system resources.
<b>1548 ERROR</b>	<b>jolt_tpset_enq failed.</b>	
	<b>Description</b>	Internal system failure.
	<b>Action</b>	Restart the client. If problem persists, check field table files and directories and then restart the servers.
<b>1549 ERROR</b>	<b>[JOLT_EVENTS failed to set %s field. Error32=%d].</b>	
	<b>Description</b>	Unable to get the field definition for TUXEDO internal fields.
	<b>Action</b>	Check TUXEDO installation and restart the servers.

<b>1550 ERROR</b>	<b>JOLT_UNSUBSCRIBE - Invalid Subscription ID.</b>	
	<b>Description</b>	Application error.
	<b>Action</b>	Check the client and restart the client.
<b>1551 ERROR</b>	<b>Memory allocation failure in JOLT_UNSUBSCRIBE.</b>	
	<b>Description</b>	Resources are not enough.
	<b>Action</b>	Increase resources and restart TUXEDO.
<b>1552 WARN</b>	<b>Dropping notification message for Transient client %d.</b>	
	<b>Description</b>	Notification arrived when a transient client is not connected.
	<b>Action</b>	Information message only; no action required.
<b>1553 WARN</b>	<b>Dropping broadcast message for Transient client %d.</b>	
	<b>Description</b>	Notification arrived when a transient client is not connected.
	<b>Action</b>	Information message only; no action required.
<b>1554 ERROR</b>	<b>Expected numeric argument for -Z option.</b>	
	<b>Description</b>	-Z option expects 0, 40, or 128 as the argument.
	<b>Action</b>	Check the configuration file and specify a valid numeric argument for JSL.
<b>1555 ERROR</b>	<b>%d - Illegal argument for -Z option.</b>	
	<b>Description</b>	Incorrect argument value is specified.
	<b>Action</b>	Check the argument for -Z option and correct it.
<b>1556 ERROR</b>	<b>%d - Illegal argument for -Z option due to international license.</b>	
	<b>Description</b>	For international release only 0 or 40 are allowed.
	<b>Action</b>	Specify correct argument.

<b>1557 ERROR</b>	<b>Incorrect number of encrypted bit values from workstation client.</b>	
	<b>Description</b>	BEA Jolt protocol failure.
	<b>Action</b>	Call BEA Technical Support.
<b>1558 ERROR</b>	<b>Expected argument to -E option.</b>	
	<b>Description</b>	An argument is expected for -E option.
	<b>Action</b>	Specify correct option and restart TUXEDO.
<b>1559 ERROR</b>	<b>%s - Illegal argument to -E option.</b>	
	<b>Description</b>	Incorrect value is specified as argument to -E option.
	<b>Action</b>	Specify the correct option.
<b>1560 ERROR</b>	<b>Cannot initialize the code conversion for local %s.</b>	
	<b>Description</b>	Cannot find function to do the code conversion for internationalization.
	<b>Action</b>	Check the shared library.
<b>1561 ERROR</b>	<b>TUXDIR is not set.</b>	
	<b>Description</b>	TUXDIR environment variable is not set.
	<b>Action</b>	Set the variable to TUXEDO directory and restart TUXEDO.
<b>1562 ERROR</b>	<b>Error reading license file.</b>	
	<b>Description</b>	Jolt is not able to open TUXEDO license file in \$TUXDIR/udataobj/lic.txt.
	<b>Action</b>	Copy the correct license file to \$TUXDIR/udataobj/lic.txt.
<b>1563 INFO</b>	<b>Serial Number: &lt;%s&gt;, Expiration Date: &lt;%s&gt;.</b>	
	<b>Description</b>	Serial number and expiration date displays.
	<b>Action</b>	No action required.

---

<b>1564 INFO</b>	<b>Licensee: &lt;%s&gt;.</b>	
	<b>Description</b>	Licensee information displays.
	<b>Action</b>	No action required.

---



# Repository Messages

<b>ERROR</b>	<b>Usage: JREPSVR [-W] -P path -W writable repository.</b>	
	<b>Description</b>	An invalid option is specified or -P is not specified properly.
	<b>Action</b>	Review the Jolt documentation and ensure that the options are specified correctly.
<b>ERROR</b>	<b>Not enough memory</b>	
	<b>Description</b>	Not enough memory; please add more swap space.
	<b>Action</b>	Configure additional memory. Make sure the operating system parameters are set correctly for the amount of memory on the machine and the amount of memory that can be used by a process. Reduce the memory usage on the machine or increase the amount of physical memory on the machine.
<b>ERROR</b>	<b>Not enough disk space for “&lt;repository-file-path&gt;”</b>	
	<b>Description</b>	Ran out of disk space while adding or deleting Repository entries, or during garbage collection.
	<b>Action</b>	Configure additional disk space.
<b>ERROR</b>	<b>Cannot modify read-only repository “&lt;repository-file-path&gt;”</b>	
	<b>Description</b>	Denies attempt to add or delete an entry from a read-only repository.
	<b>Action</b>	Check the file permission and ensure that the file is writable.
<b>ERROR</b>	<b>“&lt;repository-file-path&gt;” is not a valid repository file.</b>	
	<b>Description</b>	The specified file is not valid; a valid repository file must have the string, “#!JOLT1.0” in the first line.
	<b>Action</b>	Extract the file from the Jolt distribution CD-ROM.

<b>ERROR</b>	<b>Can't open &lt;repository-file-path&gt;.</b>	
	<b>Description</b>	Unable to open the repository file.
	<b>Action</b>	Check to ensure that the file path is valid or its permission is correct.
<b>ERROR</b>	<b>Can't create &lt;repository-file-path&gt;: check permission or path.</b>	
	<b>Description</b>	Unable to create the repository file during garbage collection.
	<b>Action</b>	Check the file or directory permission.
<b>ERROR</b>	<b>Syntax error: &lt;service definition&gt;.</b>	
	<b>Description</b>	An invalid entry was detected when an attempt was made to add an entry to the repository. The entry must have ':' as a field separator.
	<b>Action</b>	Contact BEA Technical Support.
<b>ERROR</b>	<b>Garbage collection failed: &lt;key&gt; not found.</b>	
	<b>Description</b>	When the writable repository is shutdown, it performs garbage collection to collapse the repository file. If it detects an inconsistency, the garbage collection fails.
	<b>Action</b>	Contact BEA Technical Support.

# FML Error Messages

ERROR		Fielded buffer not aligned.
	Description	An FML function was called with a fielded buffer that is not properly aligned. Most machines require half-word alignment.
	Action	Use <code>FalloC</code> to retrieve an allocated, properly aligned buffer.
	See Also	<i>TUXEDO Reference Manual</i>
ERROR		Buffer not fielded.
	Description	A buffer was passed to an FML function that has not been initialized.
	Action	Use <code>Finit</code> to initialize a buffer allocated directly by the application, or use <code>FalloC</code> to allocate and initialize a fielded buffer.
	See Also	<i>TUXEDO Reference Manual</i>
ERROR		Invalid argument to function.
	Description	An invalid argument (other than an invalid field buffer, field identifier, or field type) was passed to an FML function. This can be a parameter where a non-NULL parameter was expected (for example, it can be an invalid buffer size, etc.).
	Action	See the manual page associated with the error for the correct parameter values.
	See Also	<i>TUXEDO Reference Manual</i>

ERROR      Unknown field number or type.		
	<b>Description</b>	An invalid field number was specified for an FML function, an invalid field number (0 or greater than 8192) was specified, or <code>Fname</code> could not find the associated field identifier for the specified name.
	<b>Action</b>	Most of the FML functions return this error; see the manual page associated with the function that returned this error. Check your code to make sure the field specified is valid.
	<b>See Also</b>	<i>TUXEDO Reference Manual</i>

# Information Messages

<b>INFO</b>	<b>Repository “&lt;repository-file-path&gt;” (### records) is writable.</b>	
	<b>Description</b>	When a writable Repository server is brought up, it reports the number of records it found.
	<b>Action</b>	No action required.
<b>INFO</b>	<b>Repository “&lt;repository-file-path&gt;” (### records) is read-only.</b>	
	<b>Description</b>	When a read-only Repository server is brought up, it reports the number of records it found.
	<b>Action</b>	No action required.

# Jolt Relay Adapter (JRAD) Messages

<b>1005 ERROR</b>	<b>Memory allocation failure.</b>	
	<b>Description</b>	An attempt dynamically to allocate memory from the operating system using <code>malloc</code> failed.
	<b>Action</b>	Make sure the operating system parameters are set correctly for the amount of memory on the machine and the amount of memory that can be used by a process. Reduce the memory usage on the machine or increase the amount of physical memory on the machine. Increase the space on the swap device.
<b>1006 ERROR</b>	<b>Failed to initialize global network information.</b>	
	<b>Description</b>	The internal network information used by the JSH or the JSL was not initialized. This can happen if the system has run out of memory.
	<b>Action</b>	Increase the virtual memory available for the JSH and JSL processes.

1008 ERROR	Could not establish listening address on network.
<b>Description</b>	<p>This error occurs if the JSH or the JSL cannot advertise its listening address on the network. This could happen for one of the following conditions.</p> <ul style="list-style-type: none"><li>◆ The format of the address supplied to the JSL is incorrect. If the address format is incorrect, the network provider will be unable to advertise the address and the request fails.</li><li>◆ The address used in the <code>-n</code> command line option to the JSL is already in use by another process. For TCP/IP, this can be verified by using the <code>netstat</code> command.</li><li>◆ The system has run out of network addresses for the JSH. The JSH requests a new address from the system. If there are no addresses available, the request is rejected.</li><li>◆ A previously used address has not completed the close sequence. This occurs if the JSL or JSH was killed in an abortive manner such as <code>kill -9</code>. Some transports (among them, TCP/IP) keep the connection open for an “implementation dependent” time to flush the existing data on the buffered network connection.</li></ul>
<b>Action</b>	<p>To correct the problem, match one of the following solutions with the problem descriptions above:</p> <ul style="list-style-type: none"><li>◆ Check that the address format is correct. For TCP/IP, the format is <code>0x002ppppaaaaaaaa</code>. This is a hexadecimal representation of the TCP/IP address, where <code>pppp</code> is a unique port number and <code>aaaaaaaa</code> is the IP dotted number in the <code>/etc/hosts</code> file for the machine on which the JSL will run.</li><li>◆ See if other processes are using the requested network address. For TCP/IP, use the <code>netstat</code> command and, if the address is already in use, select a different address.</li><li>◆ If the system is out of network addresses, check with the system administrator to increase the number of addresses to use.</li><li>◆ If the connection is not closed yet, wait a few minutes and try again.</li></ul>

<b>1068 ERROR</b>	<b>Invalid command line argument '%c' ignored.</b>	
	<b>Description</b>	An illegal command line option was found in the CLOPT string.
	<b>Action</b>	Refer to the <i>BEA TUXEDO Reference Manual</i> for correct options.
<b>1074 ERROR</b>	<b>Memory allocation failure.</b>	
	<b>Description</b>	The JSL failed in an attempt to create a buffer for storing a network address.
	<b>Action</b>	<p>Make sure the operating system parameters are set correctly for the amount of memory on the machine and the amount of memory that can be used by a process.</p> <p>Reduce the memory usage on the machine or increase the amount of physical memory on the machine. Increase the space on the swap device.</p>
<b>1080 ERROR</b>	<b>Error polling network connections.</b>	
	<b>Description</b>	The JSL encountered an error polling a network connection.
	<b>Action</b>	This error indicates a network error. Check with your system administrator to see if the network is down.
<b>1081 ERROR</b>	<b>Error servicing network event.</b>	
	<b>Description</b>	The JSL was unable to process a network event.
	<b>Action</b>	This error indicates an internal problem with the JSL or the LIBNET software. Contact BEA Technical Support.



<b>1101 ERROR</b>	<b>Bad hex number provided for listening address: %s.</b>	
	<b>Description</b>	The JSL process was invoked with a <code>-n</code> option that specified a hexadecimal value as an option-argument. However, the value specified was not a valid hexadecimal constant.
	<b>Action</b>	Change the network address specified for the JSL so that it contains an even number of hexadecimal digits, and make certain that each digit is '0' through '9', 'A' through 'F', or 'a' through 'f'. Also, remember that the WSNADDR environment variable in client processes associated with this JSL must be set to this same address. The JSL <code>-n</code> option and its associated network address are part of the CLOPT parameter specified for the JSL process in the configuration file. The options for a server may be updated while the system is running through use of the <code>tmconfig (1)</code> command, or may be updated while the system is shut down by reloading the configuration file through use of <code>tmloadcf (1)</code> .

<b>1102 ERROR</b>	<b>Bad hex number provided for listening address: %s.</b>	
	<b>Description</b>	The JSL process was invoked with a <code>-n</code> option that specified a hexadecimal value as an option-argument. However, the value specified was not a valid hexadecimal constant.
	<b>Action</b>	Change the network address specified for the JSL so that it contains an even number of hexadecimal digits, and make certain that each digit is '0' through '9', 'A' through 'F', or 'a' through 'f'. Also, remember that the WSNADDR environment variable in client processes associated with this JSL must be set to this same address. The JSL <code>-n</code> option and its associated network address are part of the CLOPT parameter specified for the JSL process in the configuration file. The options for a server may be updated while the system is running through use of the <code>tmconfig (1)</code> command, or may be updated while the system is shut down by reloading the configuration file through use of <code>tmloadcf (1)</code> .

<b>1197 INFO</b>	<b>Exiting system.</b>	
	<b>Description</b>	Informational message, no action required.
	<b>Action</b>	Refer to the <i>BEA TUXEDO Administrator's Guide</i> .

<b>1202 ERROR</b>	<b>Could not initialize network.</b>	
	<b>Description</b>	An attempt to initialize the networking software from the JSL or JSH failed.
	<b>Action</b>	Make sure that the correct networking software is installed on the system and that the network is accessible.
<b>1221 ERROR</b>	<b>Unrecognized message magic %d.</b>	
	<b>Description</b>	The Jolt listener has tried all TCP ports within the range specified by the -p and -P options. It could not bind to any of the TCP ports in the range. The Jolt listener could not bind to the given address.
	<b>Action</b>	If there are more Jolt handlers than ports available in the range specified by -p and -P, then a new handler will not be able to bind to any of the TCP ports in the allowable range. Do not forget about the TCP port which is used by the workstation listener as well. Increase the range specified by the -p and -P options. Make sure that address is correct.
<b>1500 ERROR</b>	<b>Needs both -l -c options with arguments.</b>	
	<b>Description</b>	Needed options are without arguments.
	<b>Action</b>	Check and correct configuration file for JRAD entry.
<b>1501 ERROR</b>	<b>Malloc failed.</b>	
	<b>Description</b>	JRAD is not able to allocate dynamic memory.
	<b>Action</b>	Increase the system resources and restart the JRAD.
<b>1502 ERROR</b>	<b>Memory allocation failed.</b>	
	<b>Description</b>	JRAD is not able to allocate dynamic memory.
	<b>Action</b>	Increase the system resources and restart the JRAD.
<b>1503 ERROR</b>	<b>Memory allocation failed. Cannot send ESTCON.</b>	
	<b>Description</b>	JRAD is not able to allocate dynamic memory.
	<b>Action</b>	Increase the system resources and restart the JRAD.

<b>1504 INFO</b>	<b>Memory allocation failed. Cannot send ESTCON.</b>	
	<b>Description</b>	JRAD is not able to allocate dynamic memory.
	<b>Action</b>	Increase the system resources and restart the JRAD.
<b>1505 ERROR</b>	<b>Memory allocation failed. Cannot send ESTCON.</b>	
	<b>Description</b>	JRAD is not able to allocate dynamic memory.
	<b>Action</b>	Increase the system resources and restart the JRAD.
<b>1506 ERROR</b>	<b>Connection to JSL failed.</b>	
	<b>Description</b>	JSL is not running.
	<b>Action</b>	Check the address given with option -c.
<b>1507 ERROR</b>	<b>Sending message to JSL failed.</b>	
	<b>Description</b>	JSL is not running or network connection is down.
	<b>Action</b>	Restart the JRAD/JSL.
<b>1508 INFO</b>	<b>Sending message to JSH failed.</b>	
	<b>Description</b>	Network is down. Connection to the JSH failed.
	<b>Action</b>	Check the network and restart the JSL.
<b>1509 ERROR</b>	<b>Sending CONNECT reply to JRLY.</b>	
	<b>Description</b>	Unable to reach JRLY. Probably problem in the network.
	<b>Action</b>	Restart the JRLY and JRAD after check the network addresses.

<b>1510 ERROR</b>	<b>Sending SHUTDOWN reply to JRLY.</b>	
	<b>Description</b>	Unable to reach JRLY. Probably problem in the network.
	<b>Action</b>	Restart the JRLY and JRAD after check the network addresses.

# Jolt Relay (JRLY) Messages

<b>ERROR</b>	<b>Ignoring syntax error in configuration file line %d</b>	
	<b>Description</b>	The line in question doesn't contain an equal sign or (in case of the LISTEN and CONNECT tag) is missing the colon.
	<b>Action</b>	Verify the syntax of the configuration file at the specified line.
<b>ERROR</b>	<b>Ignoring unknown tag '%s' in configuration file line %d.</b>	
	<b>Description</b>	The line in question is does not contain one of the valid tags: LOGDIR, ACCESS_LOG, ERROR_LOG, LISTEN, CONNECT.
	<b>Action</b>	Verify the syntax of the configuration file at the specified line.
<b>ERROR</b>	<b>MSG_MALLOC: perror().</b>	
	<b>Description</b>	Memory allocation failed. The relay will exit.
	<b>Action</b>	Make more memory available on the machine on which the relay is running. Remove other unnecessary processes which may be running on the same host as the relay. Restart the relay.
<b>ERROR</b>	<b>Client structure != NULL for file descriptor %ld</b>	
	<b>Description</b>	An internal error occurred. The relay will continue to run, but a client process may have been disconnected.
	<b>Action</b>	None. If this message appears repeatedly and can be reproduced consistently notify BEA Technical Support.
<b>ERROR</b>	<b>Invalid file descriptor %ld</b>	
	<b>Description</b>	An internal error occurred. The relay will continue to run, but a client process may have been disconnected.
	<b>Action</b>	None. If this message appears repeatedly and can be reproduced consistently notify BEA Technical Support.

<b>ERROR</b>	<b>Could not open configuration file %s</b>	
	<b>Description</b>	The specified configuration file does not exist or is not readable. The relay will exit.
	<b>Action</b>	Check the file name and the permissions on the file and the directory.
<b>ERROR</b>	<b>No log directory specified.</b>	
	<b>Description</b>	LOGDIR was not specified in the configuration file or no value for it was given.
	<b>Action</b>	Verify the entry for the tag LOGDIR in the configuration file. Check that the correct configuration file is being used (-f parameter).
<b>ERROR</b>	<b>No access log file specified.</b>	
	<b>Description</b>	ACCESS_LOG was not specified in the configuration file or no value for it was given.
	<b>Action</b>	Verify the entry for the tag ACCESS_LOG in the configuration file. Check that the correct configuration file is being used (-f parameter).
<b>ERROR</b>	<b>No error log file specified.</b>	
	<b>Description</b>	ERROR_LOG was not specified in the configuration file or no value for it was given.
	<b>Action</b>	Verify the entry for the tag ERROR_LOG in the configuration file. Check that the correct configuration file is being used (-f parameter).
<b>ERROR</b>	<b>No JRLY host specified</b>	
	<b>Description</b>	The value for the LISTEN tag does not contain the host name or IP address or the relay host, e.g., LISTEN=host:port.
	<b>Action</b>	Verify the entry for the tag LISTEN in the configuration file. Check that the correct configuration file is being used (-f parameter).

<b>ERROR</b>	<b>No JRAD host specified.</b>	
	<b>Description</b>	The value for the CONNECT tag does not contain the host name or IP address or the JRAD host, e.g., CONNECT=host:port.
	<b>Action</b>	Verify the entry for the tag CONNECT in the configuration file. Check that the correct configuration file is being used (-f parameter).
<b>ERROR</b>	<b>No listener port specified or listener port &lt;= 0.</b>	
	<b>Description</b>	The value for the LISTEN tag does not contain a valid port number on the relay host.
	<b>Action</b>	Verify the entry for the tag LISTEN in the configuration file. Check that the correct configuration file is being used (-f parameter).
<b>ERROR</b>	<b>No JRAD port specified or JRAD port &lt;= 0.</b>	
	<b>Description</b>	The value for the CONNECT tag does not contain a valid port number on the relay host.
	<b>Action</b>	Verify the entry for the tag CONNECT in the configuration file. Check that the correct configuration file is being used (-f parameter).
<b>ERROR</b>	<b>Could not determine IP address of listener host</b>	
	<b>Description</b>	The relay could not look up the IP address of the host machine.
	<b>Action</b>	If the host was specified as a host name replace it with the IP address and restart the relay. If it already was given as IP address make sure that the IP address is correct and that you're trying to start the relay on this host. Note that the address specified must be the address of the host on which the relay is running.

ERROR		Cannot bind socket
	Description	The listener port specified in the configuration file is already being used by another application or still in a final wait state from a previous run of jrly.
	Action	Either specify a different port number in the configuration file (and all HTML files containing the IP address and port number of the relay) or wait a few minutes. The command "netstat -a" displays existing connections.
ERROR		Can't open log file %s
	Description	Either the error log file or access log file (or both) could not be opened for writing.
	Action	Check the configuration file for correct spelling of the LOGDIR. Make sure you have write permissions on this directory and the files specified. On Windows NT, the directory separators must be back slashes, not forward slashes.
ERROR		WSAStartup failed (NT only)
	Description	<p>The Winsock driver could not initialize. Possible causes:</p> <ul style="list-style-type: none"><li>◆ The underlying network subsystem is not ready for network communication Version 2.0 of Windows Sockets support is not provided by this particular Windows Sockets implementation.</li><li>◆ Limit on the number of tasks supported by the Windows Sockets implementation has been reached.</li></ul>
	Action	Check the networking software configuration on your system.
ERROR		Couldn't load Winsock Driver version 2.X. (NT only)
	Description	The relay requires Winsock version 2 or higher, but could not load it.
	Action	Check the networking software configuration on your system. An older version of Windows Sockets support was detected.



<b>ERROR</b>	<b>FATAL ERROR: unknown message code %ld.</b>	
	<b>Description</b>	Internal error. The relay will exit
	<b>Action</b>	Restart the relay. If this message appears repeatedly and can be reproduced consistently notify BEA Technical Support.
<b>ERROR</b>	<b>connect: Connection refused</b>	
	<b>Description</b>	The relay could not connect to JRAD.
	<b>Action</b>	Make sure the relay adapter (JRAD) is running. Check that the CONNECT tag in the relay configuration file identifies the correct host and port on which the JRAD is running.
<b>ERROR</b>	<b>accept(): accept failed, errno: 24, strerror: Too many open files</b>	
	<b>Description</b>	The relay tried to open more files/sockets than the system limit.
	<b>Action</b>	The default maximum number of open file descriptors for a process is 64 on most UNIX systems. Set this number to at least 1024 (with the limit or ulimit commands).

# Bulk Loader Utility Messages

<b>ERROR</b>	<b>File not found: %s</b>	
	<b>Description</b>	The specified file is not found.
	<b>Action</b>	Check the path again.
<b>ERROR</b>	<b>Error on line %d: %s value is null</b>	
	<b>Description</b>	A value is expected for this keyword.
	<b>Action</b>	Input the value.
<b>ERROR</b>	<b>Error on line %d: Invalid keyword: %s=%s</b>	
	<b>Description</b>	Keyword is not recognized.
	<b>Action</b>	Input the correct keyword value.
<b>ERROR</b>	<b>Error on line %d: Invalid number: %s</b>	
	<b>Description</b>	The numeric number is malformed.
	<b>Action</b>	Input the correct value.
<b>ERROR</b>	<b>Error on line %d: Invalid value: %s</b>	
	<b>Description</b>	The value of the parameter is out of range.
	<b>Action</b>	Input the correct value.
<b>ERROR</b>	<b>Error on line %d: Invalid value: %s</b>	
	<b>Description</b>	The data type of the parameter is invalid.
	<b>Action</b>	Input the correct value.



---

# Index



## A

- APPADDRESS 7-17
- applets
  - client-side execution 6-42
  - Java 6-1, 6-2, 6-43
  - Jolt 1-10, 6-4
  - localizing 6-44
- appletview
  - Repository Editor 5-5
- ApplicationException 7-25
  - class A-4
  - methods A-5
    - getApplicationCode A-5
    - getMessage A-5
    - getObject A-5
- applications
  - deployment 6-42
  - localization 6-42
  - multithreaded 6-21
  - sample
    - online resources 3-22

## B

- BEA TUXEDO
  - access 6-1
  - ATMI interface 6-4
  - buffer types
    - using with Jolt 6-14
  - customizing 6-1
  - data types
    - using with Jolt 6-14

- Jolt Repository Editor
  - initializing services using 3-13
- logging
  - off 6-5
  - on 6-5
- server requirements 6-42
- services
  - executing 6-5
  - requests 6-4
- transaction
  - begin 6-5
  - complete 6-5
  - new 6-5
  - rollback 6-5
- browsers
  - online documentation viewing 2-27
- buffer type
  - CARRAY 6-17
  - FML 6-19
  - STRING 6-15
  - VIEW 6-19
- buffer types
  - STRING 6-15
  - TUXEDO 6-14
- bulk loader
  - bulk load file 4-3
  - command line options 4-3
  - data file syntax 4-4
  - getting started 4-2
  - introduction 4-1
  - keywords 4-4, 4-5, 4-6, 4-8
  - messages B-29
  - sample data 4-10
  - troubleshooting 4-9
  - UNIX 4-2
  - using Windows NT 4-2

---

## C

### CARRAY

- buffer type 6-17, 6-19
- classes 6-6
  - ApplicationException A-4
  - EventException A-8
  - functionality 6-8
  - hierarchy 6-7
  - Java RuntimeException and Error A-2
  - Jolt 6-1, 6-6, 6-8
  - JoltEvent class 7-54
  - JoltException A-6
  - JoltMessage class 7-60
  - JoltRemoteService 6-8
  - JoltRemoteService class 7-23
  - JoltReply class 7-59
  - JoltRequestMessage class 7-26
  - JoltSession 6-8
  - JoltSession class 7-18
  - JoltSessionAttributes 6-6, 6-8
  - JoltSessionAttributes class 7-5
  - JoltTransaction 6-10
  - JoltTransaction class 7-50
  - JoltUserEvent class 7-56
  - MessageException A-8
  - relationships 6-7
  - ServiceException A-9
  - SessionException A-9
  - subdirectory 6-43
  - TransactionException A-10
- client
  - Jolt 6-5
  - login/logoff 6-8
- configuration 2-1, 3-1, 3-12
  - CLOPT parameters 3-7
  - command line options 3-7
  - Jolt Repository
    - \*GROUPS section 3-12
    - \*SERVERS section 3-12
  - Jolt Server Listener (JSL) 3-4

- network address 3-20
- Repository File
  - jrepository 3-12
- connection attributes 6-10
  - hostname 6-10
  - portnumber 6-10
- connection modes
  - connection-less 6-30
  - retained 6-30
- constants
  - APPADDRESS 7-17
  - IDLETIMEOUT 7-17
  - RECVTIMEOUT 7-17
  - SENDTIMEOUT 7-17
- constructors
  - JoltRemoteService constructor 7-24
  - JoltSession constructor 7-19
  - JoltSessionAttributes constructor 7-6
  - JoltTransaction constructor 7-51

## D

- data types
  - TUXEDO 6-14
- DES 1-4
- directory structure 2-4, 2-5

## E

- ECHO service parameters
  - INPUT/OUTPUT 6-17
- encryption 1-4, 3-3
- error code types A-2
- error messages
  - from tpstrerror A-2
  - getting A-2
- error number, getting A-2
- errors A-1
  - caught by Jolt A-2
  - Jolt 6-3
  - Jolt interpreter 6-3

---

- non-recoverable A-2
- summary of TUXEDO A-11
- TPEJOLT A-2
- TUXEDO A-2
- TUXEDO generated in Jolt 6-3
- Event Subscription 6-28
  - classes for 6-28
  - supported types 6-31
- event subscription 3-14
  - configuration 3-14
  - filtering buffers 3-15
- EventException class A-8
- events, subscribing to 6-28
- exceptions A-1
  - for endSession method 7-20, 7-21, 7-22
  - Jolt 6-3
  - Jolt class hierarchy A-3
  - ServiceException 6-10
  - System.in.read 6-23
- exporting services 5-38

## F

- FML buffer type 6-19

## G

- getErrno
  - using to get error number A-2
- getMessage
  - using to get error message A-2
- group services
  - package organizer, how to use 5-31
- GROUPS section configuration 3-12

## H

- HTML
  - applet tag 6-43
  - page 6-43

## I

- IDLETIMEOUT 7-17
- installation 2-1, 2-2, 2-13, 3-1
  - before you begin 2-6
  - directory structure 2-4, 2-5
  - Jolt 2-4
  - Jolt Relay 2-3
  - NT license agreement 2-22
  - online documentation 2-1, 3-1
  - requirements
    - client 2-3
    - disk storage 2-2, 2-3
    - Java Developer's Kit 2-3
    - server 2-2
  - UNIX system instructions 2-7
  - Windows NT 2-12

- Internet services 1-2
- Internet Relay 3-2, 3-16
- Intranet services 1-2

- items
  - methods for appending 7-4
  - methods for changing by index 7-3
  - methods for changing first 7-3
  - methods for getting 7-4
  - methods for handling 7-2

## J

- Java
  - applets 6-1, 6-2, 6-43
  - class files 6-43
  - clients 1-7, 6-4
  - Developer's Kit (JDK) 1.0 6-23
  - language classes 6-1
  - packages 6-43
  - programs 6-2
  - Thread.yield() method 6-22
  - Virtual Machine (VM) 6-21
  - java.lang.IllegalAccessError 7-6 — 7-16
  - java.lang.NoSuchFieldError 7-6 — 7-16

---

## Jolt

- applets 1-10
  - deploying 6-42
  - localizing 6-44
- architecture 1-3, 1-5
- bulk loader 4-1
- classes 6-1, 6-6, 6-43
  - functionality 6-8
  - hierarchy 6-7
  - relationships 6-7
  - subdirectory 6-43
- client
  - interface objects 6-5
  - login/logoff 6-8
  - populating variables 6-5
  - requests 6-5
- client/server
  - interaction 6-5
  - relationship 6-4
- clients
  - communication with servers 1-8
- connection manager 6-4
- defined 1-1, 1-2
- features 1-3
- installation
  - package 2-4
  - requirements 2-2
- international use 6-44
- Internet Relay 3-2, 3-16
- JRAD B-17
- JRLY B-24
- license 2-22
- license agreement 2-13
- Relay installation 2-3
- Repository 3-10, 6-5
  - service attributes 6-5
- Repository Editor 1-2,
  - initializing services using 3-13
  - using 5-1
- sample applications
  - simpapp 3-22

- server 3-2, 6-4, 6-5, 6-43
  - requirements 6-42
- Server Listener (JSL)
  - \*SERVERS section 3-5
  - configuration 3-5
- servers 1-2
  - communication with clients 1-8
  - components 1-6
  - proxy for TUXEDO client 1-5
  - Transaction Protocol 1-8, 6-4
  - using threads with 6-23
- Jolt Class Library 1-2, 1-7, 6-2,
  - 6-6, 6-8, 6-10, 7-1
- application development 6-42
- errors 6-3, A-1
  - handling 6-3, A-2
  - list of TUXEDO related A-11
- exceptions 6-3, A-1
  - handling 6-3, A-2
- functionality 6-8
- object/class reusability 6-35

Jolt classes

- JoltEvent 7-54
- JoltMessage 7-60
- JoltRemoteService 7-23
- JoltReply 7-59
- JoltRequestMessage 7-26
- JoltSession 7-18
- JoltSessionAttributes 7-5
- JoltTransaction 7-50
- JoltUserEvent 7-56

Jolt methods 7-2

Jolt Reply 6-28

Jolt Repository Server 1-6

Jolt Server

- shutting down the 3-10
- starting the 3-3

Jolt Server Handler 1-6

Jolt Server Listener 1-6

---

Jolt Server Listener (JSL)  
    \*MACHINES section 3-5  
    configuration 3-4  
    UBBCONFIG file 3-4  
JoltEvent 7-54  
JoltException  
    class A-3, A-6  
    methods  
        getErrno A-7  
        getMessage A-7  
        getObject A-7  
JoltMessage 6-28, 7-60  
JoltRemoteService 6-10, 7-23  
    calls 6-10  
    class 6-8  
    methods 7-24  
    object 6-8  
    resetting parameters 6-9  
    reusing 6-35  
JoltReply 7-59  
JoltRequestMessage 7-26  
    methods 7-28  
JoltSession 6-5, 6-10, 6-28, 6-33,  
    7-18  
    class 6-8, 6-10, 6-33  
    constructor 7-19  
    methods 7-20  
    object 6-7, 6-8  
        instantiating 6-10  
JoltSessionAttributes 6-6, 6-7, 6-8,  
    6-10, 7-5  
    constructor 7-6  
JoltTransaction 6-5, 6-7, 6-9, 6-10,  
    7-50  
    class 6-10  
    constructor 7-51  
    methods 7-52  
JoltUserEvent 6-28, 7-56  
JRAD 3-20  
    configuration 3-20  
    messages B-17

    network address configuration 3-20  
    starting 3-20  
jrepository 3-12  
JRLY 3-18  
    configuration 3-18  
    messages B-24  
    network address configuration 3-20  
    starting 3-18

## L

license agreement 2-13  
    installation 2-22  
licensing 2-22  
    TUXEDO 6.1, 6.2 2-23  
    TUXEDO 6.3 2-23  
logoff 6-8  
logon 6-8  
    Repository Editor 5-6

## M

MACHINES section  
    Jolt Server Listener (JSL) 3-5  
MessageException  
    class A-8  
messages  
    bulk loader B-29  
    FML B-14  
    information B-16  
    Jolt system B-2  
    JRAD B-17  
    JRLY B-24  
    repository B-12  
methods 7-2  
    ApplicationException A-5  
        getApplicationCode A-5  
        getMessage A-5  
        getObject A-5  
    clear() 6-9



---

JoltEvent			
unsubscribe	7-54		
unsubscribeAll	7-55		
JoltException			
getErrno	A-7		
getMessage	A-7		
getObject	A-7		
JoltMessage	7-61		
getBytesDef	7-61,	7-62,	
	7-63,	7-64	
getBytesItemDef	7-64		
getBytesItemDef	7-67		
getDoubleDef	7-63		
getDoubleItemDef	7-66		
getFloatItemDef	7-66		
getIntDef	7-62		
getIntItemDef	7-65		
getOccurrenceCount	7-61		
getShortItemDef	7-65		
getStringItemDef	7-67		
JoltRemoteService	7-24		
call	7-24		
JoltReply			
getMessage	7-59		
JoltRequestMessage	7-28		
addByte	7-45		
addBytes	7-46		
addDouble	7-46		
addFloat	7-47		
addInt	7-47		
addShort	7-48		
addString	7-48		
clear	7-31		
delete	7-49		
deleteItem	7-49		
getApplicationCode	7-31		
getBytesDef	7-32		
getBytesItemDef	7-35		
getBytesDef	7-32		
getBytesItemDef	7-35		
getDoubleDef	7-33		
getDoubleItemDef	7-36		
getFloatDef	7-33		
getFloatItemDef	7-36		
getIntDef	7-34		
getIntItemDef	7-36		
getName	7-31		
getOccurrences	7-32		
getShortDef	7-34		
getShortItemDef	7-37		
getStringDef	7-34		
getStringItemDef	7-37		
setByte	7-38,	7-39	
setByteItem	7-42		
setBytesItem	7-42		
setDouble	7-39		
setDoubleItem	7-43		
setFloat	7-40		
setFloatItem	7-43		
setInt	7-40		
setIntItem	7-44		
setRequestPriority	7-38		
setShort	7-41		
setShortItem	7-44		
setString	7-41		
setStringItem	7-45		
JoltSession	7-20		
endSession	7-20,	7-21	
finalize	7-22		
JoltSessionAttributes	7-7		
checkAuthenticationLevel	7-7		
clear	7-8		
getBytesDef	7-9,	7-12	
getBytesDef	7-9		
getDoubleDef	7-10		
getFloatDef	7-10		
getIntDef	7-11		
getStringDef	7-12		
global attributes for methods	7-16		
setByte	7-13		
setBytes	7-13		
setDouble	7-14		

---

- setFloat 7-14
- setInt 7-15
- setShort 7-15
- setString 7-16
- JoltTransaction 7-52
  - commit 7-52
  - rollback 7-53
- JoltUserEvent
  - JoltUserEvent 7-57
- Thread.yield() 6-22
- Microsoft Internet Explorer
  - opening documentation files 2-28
- multithreaded applications 6-21

## N

- Netscape Navigator 5-6
  - opening documentation files 2-28
- notifications
  - acknowledged 6-30
  - brokered event 6-28
  - data buffers 6-30
  - event handler for 6-29
  - unsolicited 6-28, 6-29
  - unsubscribing 6-32
  - using Jolt to receive 6-33

## O

- objects
  - relationships 6-7
  - reusability 6-28
  - reusing 6-38
- online documentation 2-27
  - browsers 2-3, 2-27
  - getting started 2-27
  - installation 2-1, 3-1
  - opening files
    - Microsoft Internet Explorer 2-28
    - Netscape Navigator 2-28
  - using 2-26

## P

- package organizer
  - description 5-31
  - group services, how to 5-31
  - using 5-29
- packages
  - add a package 5-20
  - adding 5-19
  - delete a package 5-36
  - deleting 5-37
  - modifying 5-33
  - package organizer 5-29
  - Repository Editor 5-12, 5-14
- parameters 5-17
  - delete a parameter 5-36
  - deleting 5-36
  - edit a parameter 5-36
  - editing 5-35
  - modifying 5-33

## R

- RC4 1-4
- RECVTIMEOUT 7-17
- Repository
  - configuration 3-11
- Repository Editor 1-2, 1-9
  - appletviewer 5-5
  - exiting the 5-8
  - introduction 5-2
  - login 5-6
  - main components of 5-10
  - Netscape Navigator 5-6
  - packages 5-12, 5-14
    - setting up 5-19
  - parameters 5-17
  - process flow 5-10
  - sample window 5-3
  - sample window description 5-4
  - saving your work 5-19

---

- services 5-15
  - description of 5-16
  - setting up 5-19
  - view services 5-16
- troubleshooting 5-48
- requirements 2-2, 2-27

## S

- saving your work 5-19
- security 1-4, 3-3
- SENDTIMEOUT 7-17
- server
  - installation requirements
    - AIX 2-2
    - disk storage 2-2
    - Hewlett-Packard 2-2, 2-3
  - Jolt 6-5
  - TUXEDO requirements for 6-42
  - web 6-43
- servers
  - components 1-6
  - Jolt 1-2
  - Jolt Repository 1-6
- ServiceException
  - class A-9
- ServiceExceptions 7-25
- services
  - add a parameter 5-25
    - data type selection 5-28
    - how to 5-27
    - window description 5-26
  - add a service 5-21
    - buffer type selection 5-23
    - how to 5-22, 5-23
  - calling synchronous 6-8
  - definitions 6-11
  - delete a service 5-36
  - deleting 5-37
  - edit a service 5-33
  - editing 5-34

- export status
  - reviewing 5-40, 5-41
- exporting 5-38, 5-39
- grouping 5-29
- Internet 1-2
- Intranet 1-2
- Jolt client
  - make service available to 5-38
- modifying 5-33
- parameters 5-17
- service test window 5-43, 5-44
- test a service
  - failure, reasons for 5-47
  - how to 5-45, 5-46
  - process flow 5-45
- testing 5-42
- unexport 5-38
- unexport a service 5-39
- unexport status
  - reviewing 5-40, 5-41
- using the Repository Editor 5-15
- view parameters 5-18
- view services 5-16
- SessionException 7-19
  - class A-9
- simpapp
  - online resources 3-22
- STRING buffer type 6-15

## T

- testing
  - services 5-42
- threads
  - BLOCKED 6-21
  - non-preemptive 6-21, 6-22
  - preemptive 6-21
  - RUNNABLE 6-21
  - RUNNING 6-21
  - using Jolt with non-preemptive 6-22
  - using with Jolt 6-23

---

## TOUPPER

input parameters 6-15  
service 6-15

TPEABORT A-11

TPEBADDESC A-11

TPEBLOCK A-11

TPEDIAGNOSTIC A-11

TPEEVENT A-11

TPEHAZARD A-11

TPEHEURISTIC A-11

TPEINVAL A-11

TPEITYPE A-11

TPELIMIT A-11

TPEMATCH A-11

TPEMIB A-11

TPENOENT A-11

TPEOS A-11

TPEOTYPE A-11

TPEPERM A-12

TPEPROTO A-12

TPERELEASE A-12

TPERMERR A-12

TPESVCERR A-12

TPESVCFAIL A-12

TPESYSTEM A-12

TPETIME A-12

TPETRAN A-12

TPGOTSIG A-12

tpreturn A-4

tpurcode A-5

Transaction

begin 6-9

commit 6-9

object 6-9

Protocol 6-4

rollback 6-9

TransactionException

class A-10

troubleshooting

Repository Editor 5-48

## TUXEDO

distributing services 1-9  
errors A-11

## U

### UBBCONFIG

Jolt Repository configuration sample  
3-11

Jolt Server Listener (JSL) configuration  
sample 3-4

unexporting services 5-38

UNIX system

installation 2-7

UNSOLMSG 7-56

## V

VIEW buffer type 6-19

view parameters 5-18

## W

web server

considerations 6-43

Windows NT

installation 2-12

