SYBASE®

Automated Configuration Guide

EAServer

6.0

DOCUMENT ID: DC00548-01-0600-01

LAST REVISED: July 2006

Contents

About This Book

Subject

This book describes the automated configuration features available in EAServer.

Audience

This book is for administrators and application providers that must perform scripted configuration to automate deployment of application components and server configurations.

How to use this book

Chapter 2, “Ant-Based Configuration,” describes how to manage application servers, components, and resources using the EAServer Ant based configuration mechanism.

Chapter 3, “Using Scheduled Tasks,” describes how to create and run scheduled tasks to automate periodic maintenance such as checking log files or running database cleanup commands.

Chapter 4, “Creating Service Components,” describes how to create service components, which run a specially coded application component automatically in the server as a background task.

Chapter 5, “Using the Thread Manager,” describes how to use the Thread Manager to to start threads from EAServer components to perform asynchronous processing.

Chapter 6, “Using jagtool and jagant,” contains information about these command line tools that allow you to automate some EAServer development and deployment tasks.

Related documents

Core EAServer documentation The core EAServer documents are available in HTML and PDF format in your EAServer software installation and on the SyBooks™ CD.

What’s New in EAServer 6.0 summarizes new functionality in this version.

The *EAServer API Reference Manual* contains reference pages for proprietary EAServer Java classes and C routines.

The *EAServer Automated Configuration Guide* (this book) explains how to use Ant-based configuration scripts to:

- Define and configure entities, such as EJB modules, Web applications, data sources, and servers

- Perform administrative and deployment tasks

The *EAServer CORBA Components Guide* explains how to:

- Create, deploy, and configure CORBA and PowerBuilder™ components and component-based applications
- Use the industry-standard CORBA and Java APIs supported by EAServer

The *EAServer Enterprise JavaBeans User's Guide* describes how to:

- Configure and deploy EJB modules
- Develop EJB clients, and create and configure EJB providers
- Create and configure applications clients
- Run the EJB tutorial

The *EAServer Feature Guide* explains application server concepts and architecture, such as supported component models, network protocols, server-managed transactions, and Web applications.

The *EAServer Java Message Service User's Guide* describes how to create Java Message Service (JMS) clients and components to send, publish, and receive JMS messages.

The *EAServer Migration Guide* contains information about migrating EAServer 5.*x* resources and entities to an EAServer 6.0 installation.

The *EAServer Performance and Tuning Guide* describes how to tune your server and application settings for best performance.

The *EAServer Security Administration and Programming Guide* explains how to:

- Understand the EAServer security architecture
- Configure role-based security for components and Web applications
- Configure SSL certificate-based security for client connections
- Implement custom security services for authentication, authorization, and role membership evaluation
- Implement secure HTTP and IIOP client applications
- Deploy client applications that connect through Internet proxies and firewalls

The *EAServer System Administration Guide* explains how to:

- Start the preconfigured server and manage it with the Sybase Management Console
- Create, configure, and start new application servers
- Define database types and data sources
- Create clusters of application servers to host load-balanced and highly available components and Web applications
- Monitor servers and application components
- Automate administration and monitoring tasks with command line tools

The *EAServer Web Application Programming Guide* explains how to create, deploy, and configure Web applications, Java servlets, and JavaServer Pages.

The *EAServer Web Services Toolkit User's Guide* describes Web services support in EAServer, including:

- Support for standard Web services protocols such as Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Uniform Description, Discovery, and Integration (UDDI)
- Administration tools for deployment and creation of new Web services, WSDL document creation, UDDI registration, and SOAP management

The *EAServer Troubleshooting Guide* describes procedures for troubleshooting problems that EAServer users may encounter. This document is available only online; see the EAServer Troubleshooting Guide at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.eas_5.2.eastg/html/eastg/title.htm.

jConnect for JDBC documents EAServer includes the jConnect™ for JDBC™ 6.0.5 driver to allow JDBC access to Sybase database servers and gateways. The *jConnect for JDBC 6.0.5 Programmer's Reference* is available on the Sybase Product Manuals Web site at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.jconnjdbc_6.05.prjdbc/html/prjdbc/title.htm&toc=/com.sybase.help.jconnjdbc_6.05/toc.xml.

Sybase Software Asset Management User's Guide EAServer includes the Sybase Software Asset Management license manager for managing and tracking your Sybase software license deployments. The *Sybase Software Asset Management User's Guide* is available on the Getting Started CD and in the EAServer 6.0 collection on the Sybase Product Manuals Web site at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.eas_6.0/title.htm.

Conventions

The formatting conventions used in this manual are:

Formatting example	To indicate
commands and methods	When used in descriptive text, this font indicates keywords such as: • Command names used in descriptive text • C++ and Java method or class names used in descriptive text • Java package names used in descriptive text • Property names in the raw format, as when using Ant or jagtool to configure applications rather than the Management Console
variable, *package*, or *component*	Italic font indicates: • Program variables, such as *myCounter* • Parts of input text that must be substituted, for example: `Server.log` (with *Server* in italics) • File names • Names of components, EAServer packages, and other entities that are registered in the EAServer naming service
File | Save	Menu names and menu items are displayed in plain text. The vertical bar shows you how to navigate menu selections. For example, File | Save indicates "select Save from the File menu."
`package 1`	Monospace font indicates: • Information that you enter in the Management Console, a command line, or as program text • Example program fragments • Example output fragments

Other sources of information

Use the Sybase Getting Started CD, the SyBooks CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

  Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.

- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Sybase Product Manuals Web site, go to Product Manuals at http://sybooks.sybase.com/nav/base.do.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **Finding the latest information on product certifications**

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Select Products from the navigation bar on the left.
3. Select a product name from the product list and click Go.
4. Select the Certification Report filter, specify a time frame, and click Go.
5. Click a Certification Report title to display the report.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ **Finding the latest information on EBFs and software maintenance**

1. Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.
2. Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
3. Select a product.

4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

   Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the "Technical Support Contact" role to your MySybase profile.

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Accessibility features

EAServer has been tested for compliance with U.S. government Section 508 Accessibility requirements. The online help for this product is also provided in Eclipse help formats, which you can navigate using a screen reader.

The Web console supports working without a mouse. For more information, see "Keyboard navigation" in Chapter 2, "Management Console Overview," in the *EAServer System Administration Guide*.

The Web Services Toolkit plug-in for Eclipse supports accessibility features for those that cannot use a mouse, are visually impaired, or have other special needs. For information about these features see the Eclipse help:

1 Start Eclipse.

2 Select Help | Help Contents.

3 Enter `Accessibility` in the Search dialog box.

4 Select Accessible User Interfaces or Accessibility Features for Eclipse.

Note You may need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For additional information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

Introduction

EAServer provides several configuration tools and programming interfaces that allow you to automate installation, deployment, and maintenance tasks.

Silent installation

On most platforms, the EAServer installation media includes a silent installation option. You can configure a silent install kit to automate a scripted installation of the software. For details, see the *EAServer Installation Guide* for your platform.

Command-line configuration tools

EAServer provides many command-line tools that can be called from operating system scripts to automate configuration and maintenance. For more information, see Chapter 12, “Command Line Tools,” in the *System Administration Guide*.

Ant-based configuration

EAServer includes built-in Ant support and uses Ant build files to perform many administrative and deployment tasks. You can run Ant configuration scripts to define and configure entities such as EJB modules, Web applications, data sources, servers, and so forth. You can also embed Ant configuration scripts in J2EE deployment archives such as an application EAR file. The script runs automatically when you deploy the archive to EAServer. For more information, see Chapter 2, "Ant-Based Configuration."

Scheduled tasks

In EAServer, a *scheduled task* is a background task that runs at server startup, shutdown, or at scheduled intervals during server operation. You can use scheduled tasks to automate periodic configuration and maintenance activities.EAServer provides several predefined tasks to perform common maintenance operations such as collecting memory and component invocation statistics. You can define new tasks by assembling them from common, predefined operations such as checking log files and sending administrator mail. You can also define tasks to call application code, such as an enterprise JavaBean method or an application client. For more information, see Chapter 3, "Using Scheduled Tasks."

Service components

Service components allow you to run specially coded application components as a service in the server. Service components load at startup and run independently of client interaction. You can use service components to perform background processing or to provide common services for EAServer clients and other EAServer components. For more information, see Chapter 4, "Creating Service Components."

The Thread Manager

The Thread Manager allows you to spawn threads from application code running in the server. Like service components, you can use the Thread Manager to run code in the background. However, the Thread Manager allows you to start a background task programmatically. For more information, see Chapter 5, “Using the Thread Manager.”

Jagtool and jagant

The jagtool and jagant command line tools are provided primarily for backward compatibility. These tools allow you to perform common configuration options from operating system scripts or from an Ant build file. Chapter 6, “Using jagtool and jagant,” describes these tools.

CHAPTER 2 Ant-Based Configuration

EAServer includes built-in Ant support and uses Ant build files to perform many administrative and deployment tasks. You can run Ant configuration scripts to define and configure entities such as EJB modules, Web applications, data sources, servers, and so forth.

EAServer generates Ant configuration scripts when you deploy J2EE archives. The generated script applies settings from the archive's deployment descriptor to the new EAServer entities. You can also embed a user configuration file in J2EE archives to configure settings beyond those that can be specified by the deployment descriptor.

Entity types that are not deployed in J2EE archives can also be created or managed with configuration scripts. For example, you can create data sources, restart servers, and manage security roles and domains.

Topic	Page
About Ant	6
Ant in EAServer	6
EAServer configuration targets	7
Configuration scripts for J2EE modules	9
User-configuration scripts	10
Applying settings from configuration scripts	12
Predefined configuration scripts	13
Structure of a configuration script	15
Ant configuration command syntax	16
Examples	16

About Ant

Ant is an open-source, Java-based build tool provided by the Apache Software Foundation. Like UNIX make, Windows nmake, and other build tools, Ant can be used to automate repetitive tasks such as building software, deploying Web sites, and configuring application servers. Unlike make, nmake, and other shell-based build tools, Ant build commands use a platform-independent XML format.

Ant build files use XML syntax consisting of **targets** and **tasks**. Targets define the sequence of commands to build a deliverable or achieve a specific outcome; for example: compiling Java source files or rebooting a server. Targets have a name that allowed them to be invoked from the Ant command line. For example, this command invokes the target named `create-ejbjar` in the build file *mybuild.xml*:

```
ant -f mybuild.xml create-ejbjar
```

The build sequence in each target is defined by calling Ant tasks. Tasks perform a step in the build process, such as invoking the Java compiler, copying or deleting files, creating a Java archive, or running an XSLT transform. Tasks are implemented as Java classes. Ant includes many built-in tasks and EAServer provides additional configuration tasks.

For more information, see the Ant documentation on the Apache Ant Web site at http://ant.apache.org/.

Ant in EAServer

EAServer runs Ant build files to perform most administrative and deployment tasks. You can run Ant configuration scripts to define and configure entities such as EJB modules, Web applications, data sources, servers, and so forth.

When you deploy J2EE archives, EAServer generates Ant configuration files to apply the configuration settings described in the J2EE deployment descriptor. You can embed EAServer user configuration files to customize the settings and perform configuration tasks beyond those described in the deployment descriptor, such as defining data sources or administrative user accounts.

EAServer includes the following utilities to run Ant build files:

- djc-ant – to run the included Ant installation in the EAServer environment.

- configure – to run Ant and invoke the `configure` target in an EAServer configuration file
- recompile – to run Ant and invoke the `recompile` target in an EAServer configuration file
- jagant – which is the same as djc-ant and provided for backward compatibility with EAServer 5.*x*

EAServer configuration targets

EAServer defines the standard configuration targets in Table 2-1 to perform common deployment and setup tasks. Each Ant build file that is associated with a J2EE deployment archive can define these targets.

Table 2-1: EAServer configuration targets

Target	Description
The deploy target	Invoked immediately after deploying a J2EE archive, and before invoking the `configure` target. Executes one time only.
The configure target	Creates and configures entities, setting properties, assigning users to roles, creating user accounts, and so forth.
The recompile target	Executes the `configure` task (via an Ant dependency), then regenerates runtime classes required to integrate the entity implementation files into EAServer.
The refresh target	Reloads the entity implementation classes and applies property changes.
The undeploy target	Invoked before undeploying entities deployed from a J2EE archive. This target executes one time only.

The deploy target

EAServer executes this target one time only after you deploy a J2EE archive. You can use this target to perform tasks that must be done after deployment but that should not execute again whenever the `configure` and `recompile` targets are invoked. For example, to perform one-time setup of a runtime database or to create an administrator login account.

To add your own tasks that execute as part of the `deploy` target, embed a Sybase user configuration file in the J2EE archive. Inside that build file, run the tasks in a target named `deploy-user`.

The configure target

EAServer invokes this target after deploying a J2EE archive, when you reconfigure or recompile J2EE modules in the Management Console, and when you run the configure or recompile utilities. Typically, the target creates and configures entities, setting properties, assigning users to roles, creating user accounts, and so forth.

To add your own tasks to a J2EE entity's configuration, create a user configuration file for the entity and define a target named `configure-user`.

The `configure` target can also be used in build files that are not associated with J2EE archives. The predefined Ant build files use this target to create entities that are not deployed from J2EE archives—see "Predefined configuration scripts" on page 13.

The recompile target

EAServer invokes this target after deploying a J2EE archive, when you recompile J2EE modules in the Management Console, and when you run the recompile utility. This target regenerates runtime classes required to integrate the entity implementation files into EAServer. Since this target depends on the `configure` target, the `configure` tasks execute whenever you run the `recompile` target.

To add your own tasks to a J2EE entity's recompile target, create a user configuration file for the entity and define a target named `recompile-user`.

The refresh target

EAServer invokes this target after deploying a J2EE archive, when you select Refresh for entity's context menu in the Management Console, and when you run the refresh utility. This target reloads the entities implementation files into EAServer and applies property changes that affect runtime behavior, such as changing or adding JNDI name bindings.

If you have run the `configure` or `recompile` target on an entity, you must run the `refresh` target so that changes to the generated code and properties take affect.

To add your own tasks to a J2EE entity's refresh target, create a user configuration file for the entity and define a target named `refresh-user`.

The undeploy target

EAServer executes this target one time only when you undeploy a J2EE archive. The target executes when undeploy with the Management Console or the undeploy utility. You can use this target to undo configuration done in the `deploy` target.

To add your own tasks that execute as part of the `undeploy` target, embed a Sybase user configuration file in the J2EE archive. Inside that build file, run the tasks in a target named `undeploy-user`.

Configuration scripts for J2EE modules

When you deploy a J2EE archive with the Management Console or the deploy utility, EAServer generates an Ant configuration script that creates the corresponding EAServer entities and configures them with settings read from the deployment descriptor.

Configuration scripts reside in the EAServer *config* subdirectory with a naming prefix based on the EJB archive type. Table 2-2 lists the configuration file names for each J2EE entity type. The default configuration contains settings that match the deployment descriptor. You can define user configuration files to override the default settings, or to perform configuration tasks beyond those that can be described in the deployment descriptor.

Table 2-2: Configuration file names for J2EE entities

J2EE entity type	Default configuration file	User configuration file
Application	`application-`*`name`*`.xml`	`application-`*`name`*`-user.xml`
Application client	`appclient-`*`name`*`.xml`	`appclient-`*`name`*`-user.xml`
Connector or resource adaptor	`connector-`*`name`*`.xml`	`connector-`*`name`*`-user.xml`
EJB module (contains components deployed from a single EJB-JAR file)	`ejbjar-`*`name`*`.xml`	`ejbjar-`*`name`*`-user.xml`
Web application	`webapp-`*`name`*`.xml`	`webapp-`*`name`*`-user.xml`
CORBA package	`corba-`*`name`*`.xml`	`corba-`*`name`*`-user.xml`

In Table 2-2, *name* represents the base name of the archive file from which the entities are deployed. For example, if you deploy EJB components from *ejb1.jar*, the corresponding configuration file is *config/ejbjar-ejb1.xml*. Deploying *ejb1.jar* creates a package that contains the components defined in the JAR file, and the configuration file contains settings to configure and generate code for the package and all the components inside it.

Note Do not edit generated configuration scripts. Changes you make to the main configuration script are overwritten if you redeploy the module. Customize the component properties by creating a user configuration script with settings that override those in the main script.

When you redeploy entities, existing configuration scripts are backed up to the config/old directory, with numeric suffixes appended to the file name. For example, *ejbjar-foo.xml* is backed up as *ejbjar-foo.xml.1*, *ejbjar-foo.xml.2*, and so forth. EAServer retains up to ten previous backups.

User-configuration scripts

To customize the entity properties or component attributes in a module deployed from a J2EE archive, you can create a user-configuration script, which is called when you redeploy or reconfigure the module. You can also embed user configuration scripts in J2EE archives, to configure EAServer deployment settings that cannot be specified by deployment descriptor settings, such as creation of new security roles or binding components external to an EJB-JAR to EJB naming references in the module.

When you reconfigure or recompile entities deployed from J2EE modules, EAServer first calls the default (generated) Ant configuration script, then calls the user-configuration script. Therefore, settings in the user-configuration script override those in the Ant configuration script.

For J2EE entities, the user configuration script has the same base name as the generated script listed in Table 2-2, with `-user` appended. For example, if the generated script is *ejbjar-ejb1.xml*, the user configuration script is *ejbjar-ejb1-user.xml*.

You also use user-configuration scripts to automate the management of entities that are not deployed from J2EE archives such as data sources or JMS message queues.

Creating user-configuration scripts

User configuration scripts can be created manually or using the Management Console. To create scripts manually, use a text editor, an XML editor, or an IDE that understands Ant build files to create the script in the EAServer *config* subdirectory. See "Structure of a configuration script" on page 15 for details.

The procedures below describe how to create scripts in the Management Console.

❖ Creating a user-configuration script for entities deployed from J2EE archives

To create a user-configuration script for a deployed application, EJB, or Web application module:

1. In the Management Console, select the module, right-click, and select Create User Configuration. A user-configuration script is created.
2. Right-click again, and select Refresh. The user-configuration script displays on the User Configuration tab.
3. Edit the values you want to change in the script. Click Apply to save your changes.
4. For the changes to take affect, perform the following:
   a. Recompile the module by highlighting the module, right-clicking, and selecting Run Ant Recompile.
   b. Refresh the module by highlighting the module, right-clicking, and selecting Refresh. EAServer applies the property changes and refreshes the entity implementations so the new behavior takes affect.

❖ Creating configuration scripts for entities of other types

All configuration files defined in the EAServer installation appear under the Configuration Files folder in the Management Console. You can create generic configuration files here to manage entities of any type. Create them as follows:

1. In the Management Console, right-click the Configuration Files folder and choose Add.
2. Run the Add wizard and specify a name for the new file.

Embedding configuration scripts in J2EE archives

You can embed user configuration scripts in J2EE archives to automate post-deployment configuration. When you deploy the archive to EAServer, EAServer copies the script into the installation as a user configuration script. If present, the `deploy-user`, `configure-user`, and `recompile-user` targets execute after deployment. The `undeploy-user` target executes when you undeploy the module.

Place the configuration file in the archive's *META-INF* subdirectory, using the file name that matches the archive type, as listed in the second column of Table 2-3.

Table 2-3: File names for Sybase configuration files in J2EE archives

Archive type	File name in archive
J2EE application	*sybase-application-config.xml*
EJB JAR	*sybase-ejbjar-config.xml*
Web application	*sybase-webapp-config.xml*
J2EE application client	*sybase-client-config.xml*
Connector	*sybase-connector-config.xml*

After deployment, EAServer copies the configuration file to the *config* subdirectory of the EAServer installation, and renames it using the standard naming scheme for user configuration files–see Table 2-2 on page 9.

Applying settings from configuration scripts

You can configure or recompile entities in the Management Console or by using command-line tools.

❖ **Updating configurations with the Management Console**

1 Start the Management Console and connect to EAServer as described in Chapter 2, "Management Console Overview," in the *System Administration Guide*.

2 In the left frame, right-click the icon for the configuration file or entity and choose one of the following:

- Run Ant Configure, to apply the XML configuration file to the package and component properties.

- Run Ant Recompile, to apply the XML configuration file to the package and component properties and recreate generated classes.

3 If you have modified a Web application or EJB module, you must refresh the module for the changes to take affect. In the left frame, right-click the icon for the entity and choose Refresh.

❖ **Updating configurations with the command line**

1 If necessary, use a text or XML editor to edit the contents of the configuration file.

2 Run the recompile utility to apply the changes to the component and recreate generated classes. See Chapter 12, “Command Line Tools,” in the *New Features Guide* for details on this utility.

You can also run the configure utility to update the component properties without regenerating affected code.

3 If you have modified a Web application or EJB module, you must refresh the module by running the refresh utility.

Predefined configuration scripts

The installation includes default configurations for several module types, including data sources, database types, EJB providers, export configurations, and thread monitors. Using the Management Console, you can view the predefined Ant scripts by expanding the Configuration Files folder, and selecting the file; the contents of the file display in the right pane.

Table 2-4 lists the predefined scripts and what they configure.

Table 2-4: Predefined configuration scripts

Name	Configures defaults for
camel-case-off.xml	Defines a configure target to turn off camel-case style mappings of IDL types to Java types.
camel-case-on.xml	Defines a configure target to enable camel-case style mappings of IDL types to Java types.
default-application-servers.xml	New application server instances.
default-code-sets.xml	Character sets present in the server.
default-database-types.xml	Database types. See Chapter 4, “Database Access,” in the *System Administration Guide*.

Name	Configures defaults for
default-data-sources.xml	Data sources. See Chapter 4, “Database Access,” in the *System Administration Guide*.
default-ejb-providers.xml	EJB providers. See Chapter 3, “Developing EJB Clients,” in the *EJB Users Guide*.
default-export-configurations.xml	Export configurations. See Chapter 7, “Exporting Server Modules,” in the *System Administration Guide*.
default-http-contexts.xml	HTTP contexts for Web applications.
default-install.xml	Entities configured with input gathered by the installation program, such as port numbers for the default network listeners.
default-jms-providers.xml	JMS providers.
default-jms-resources.xml	Sample JMS resource factories.
default-mail-sessions.xml	JavaMail sessions.
default-scheduled-tasks.xml	Scheduled tasks.
default-security-domains.xml	Security domains.
default-security-profiles.xml	Security profiles.
default-service-components.xml	Service components.
default-system-components.xml	System components such as the message service and deployment service.
default-socket-listeners.xml	Network listeners.
default-transaction-batches.xml	Transaction batching configurations.
default-thread-monitors.xml	Thread monitors.
default-windows-service.xml	Installation of EAServer as a Windows service.
default-windows-service-sybmaster.xml	Installation of sybmaster, the EAServer SNMP master agent, as a Windows service.
deploy-tool-options.xml	Options for the deploy utility.
idl-generator-off.xml	Turns off generation of IDL for EJB components deployed from EJB-JAR files.
idl-generator-on.xml	Turns on generation of IDL for EJB components deployed from EJB-JAR files.
idl-style-cts.xml	Configures the IDL generator to create “CTS” style IDL. This style is recommended when clients are primarily PowerBuilder. It also provides backward compatibility with EAServer 5.*x* clients.
idl-style-xdt.xml	Configures the IDL generator to create “XDT” style IDL. This style is recommended when clients are primarily C++.
jacc-provider-info.xml	Example settings to install a customized JACC provider.
long-transactions-off.xml	Disables support for long transactions.
long-transactions-on.xml	Enables support for long transactions.

Structure of a configuration script

If you manually create configuration scripts, follow the structure described below. You can use an existing script as a template. For J2EE archives that have been deployed already, you can use the Management Console to create a user configuration script template as described in “Creating user-configuration scripts” on page 11.

Scripts must be run from the EAServer *config* subdirectory. Otherwise, Ant imports are not resolved, and integration with the Management Console and command-line tools will not work.

Add required imports

Configuration scripts must import *ant-config-tasks.xml*:

```
<import file="ant-config-tasks.xml"/>
```

Add the imports inside the project tag, above any target definitions.

Optionally add Ant property definitions

In the default (generated) configuration file for a J2EE archive, settings that may be used in multiple places are defined as Ant properties at the top of the default configuration file. For example, since all components deployed from the same archive typically connect to the same database, EAServer defines a `sql.dataSource` property as follows:

```
<property name="sql.dataSource" value="default"/>
```

This property specifies the data source name that is bound to any resource reference names used in the package.

In the user configuration file, you can override the definitions of these properties by redefining them. However, if your script imports the default configuration file, you must define the property in the user configuration file before the import of the default configuration file. For example:

```
<property name="sql.dataSource" value="Glossary"/>
<import file="ant-config-tasks.xml"/>
<import file="ejbjar-ejbtut.xml"/>
```

In Ant, the first declaration of a property takes precedence. If you define the property after the import of the default configuration file, the value in the default configuration takes precedence.

Define required targets

If you are creating a user configuration file for a J2EE archive, define any of the `deploy-user`, `configure-user`, `recompile-user`, and `undeploy-user` targets. "EAServer configuration targets" on page 7 describes what these targets should do. You do not need to define all targets. EAServer ignores missing target names.

If you are creating a configuration file that is not associated with a J2EE archive, define a `configure` target.

Ant configuration command syntax

For details on the syntax of Ant commands, see the following file in your EAServer installation:

```
html\help\en\index.html
```

Examples

For examples of configuration targets, see the *default-*.xml* configuration scripts in the *config* subdirectory of your installation. These scripts create the predefined entities in the installation. You can use them as a template to define and configure new entities.

The Enterprise Java Beans tutorial shows how to embed a user-configuration in an EJB-JAR file to run configuration commands at deployment time. To run the tutorial, see Chapter 6, "Tutorial: Creating Enterprise JavaBeans Components and Clients," in the *Enterprise JavaBeans User's Guide*.

CHAPTER 3 Using Scheduled Tasks

About scheduled tasks

A scheduled task is a background task that runs at server startup, shutdown, or at scheduled intervals during server operation. You can use scheduled tasks to automate periodic configuration and maintenance activities.

EAServer provides several predefined tasks to perform common maintenance operations such as collecting memory and component invocation statistics.

You can define new tasks to run predefined task operations, such as:

- Parsing logs for errors
- Running SQL commands through a data source to configure the remote database server
- Posting a message to a JMS message queue
- Calling application code such as an EJB stateful session bean method or running an application client
- Running operating system commands
- Sending email to server administrators

You can also assemble task chains, to combine operations defined in multiple tasks. For example, you can combine tasks that check for log errors with an email task to alert administrators to errors in the log.

Service components can also run automated tasks using code that you write yourself.—see Chapter 4, "Creating Service Components." EAServer activates scheduled tasks after service components. A scheduled task configured to run at server startup runs after services are started.

Predefined tasks

Table 3-5 describes the tasks that you can schedule to be performed automatically. To schedule tasks individually, choose Select, then select or unselect each task. To schedule all the tasks, select All.

To add new tasks and configure task schedules, see "Creating new tasks" on page 19.

Table 3-1: Scheduled tasks

Task	Description
AutoDeploy	Automatically deploy any EJB-JAR, WAR, EAR, or application-client JAR that is in the *deploy* directory. By default, this is enabled.
AutoRefresh	Enables EJBs and Web modules that are refreshed by AutoDeploy, or by running the deploy or refresh commands to be reloaded by the server. By default, this is enabled.
CheckForApplicationExceptions	Check the log for application exceptions that match a given string.
CheckForErrorMessages	Check the log for error messages that match a given string.
CheckForSecurityAlerts	Check the log for security alerts that match a given string.
CheckForSystemExceptions	Check the log for system exceptions that match a given string.
CheckForWarningMessages	Check the log for warning messages that match a given string.
CheckMemoryUsage	Compare memory usage to the memory monitor limits configured in the server properties. For more information on memory limits, see "Monitors tab" in Chapter 3, "Creating and Configuring Servers," in the *System Administration Guide*.
Dump60MinuteMemoryUsage	Write 60-minute memory usage statistics to the log file every hour.
Dump60MinuteStatistics	Write 60-minute statistics to the log file every hour. Files are written in the *logs/statistics* directory.
Dump60SecondMemoryUsage	Write 60-second memory usage statistics to the log file every minute.
Dump60SecondStatistics	Write 60-second statistics to the log file every minute. Files are written in the *logs/statistics* directory
EjbSessionTimeout	Deletes session state from the session.db data source for passivated EJB stateful session bean instances that have timed out due to inactivity.

Task	Description
EndOfServerLog	Used by other tasks; for example, to include part of the server log in an e-mail message.
JmsDuplicateDetectionExpiry	Deletes expired JMS duplicate-detection records.
JmsPersistentMessageExpiry	Deletes expired JMS persistent messages.
LocalRestart	Enables the -local option for restarting the local server.
LocalStop	Enables the -local option for stopping the local server.
NotifyApplicationExceptions	Send a notification if application exceptions are logged.
NotifyErrorMessages	Send a notification if errors are logged.
NotifySecurityAlerts	Send a notification if a security violation occurs.
NotifyServerShutdown	Send a notification when the server shuts down.
NotifyServerStartup	Send a notification when the server starts.
NotifySystemExceptions	Send a notification if system exceptions occur.
NotifyWarningMessages	Send a notification if warnings are logged.
PbHeap_dumpSummary	Print a summary of the PowerBuilder heap memory usage.
StartOfServerLog	Used by other tasks; for example, to include part of the server log in an e-mail message.
SybHeap_dumpSummary	Print a summary of the EAServer heap memory usage.
TxRef	Log transaction cross references. To run this task, the server -txRef option must be set to true.
XaRecovery	Recover transactions that did not complete due to server failure. This option sends a recovery call to the database to get the list of transaction IDs, then either commits or rolls back each transaction, depending on its state.

Creating new tasks

You can create scheduled tasks with the Management Console or by running an Ant configuration script.

The scheduled task properties specify which actions to perform and when to run the tasks.

❖ Creating or modifying scheduled tasks with the Management Console

1 In the Management Console, expand the Scheduled Tasks folder.

2 To modify an existing task, highlight the task to configure.

To create a new task, right-click the Scheduled Tasks folder and choose Add from the context menu. The Add wizard runs. Specify the task name when prompted for it and click Finish.

3. The tasks properties display on tabbed pages in the right pane. Set the scheduled task options using these pages. Click Apply to save changes before moving between tabs.

   Properties are described in "Task properties on the General tab" on page 20 and "Task properties on the General tab" on page 20.

4. To run the task, follow the instructions in "Configuring scheduled tasks to run" on page 25

❖ **Creating or modifying scheduled tasks using Ant**

1. In the EAServer *config* subdirectory, define an Ant configuration script with a configure target that calls the <setProperties> command to create an entity of type scheduledTask. For an example, see "Ant configuration example" on page 24.

   The Ant property names are listed in Table 3-2, Table 3-3, and Table 3-4. Ant property names are provided in parentheses after the Management Console display name. Properties are also described in the HTML reference located in the following file in your installation:

   ```
   html\help\en\com.sybase.djc.scheduler.ScheduledTask
   .html
   ```

2. Run the script with the configure command-line tool, specifying the base name of the configuration script as the argument.

3. To run the task, follow the instructions in "Configuring scheduled tasks to run" on page 25.

Task properties on the General tab

The properties on the General tab, specify which tasks to perform. Table 3-2 on page 21 defines the options that are common to all the tasks. Ant property names are provided in parentheses after the Management Console display name.

Table 3-2: Scheduled task common options

Property	Description
Task Action (taskAction)	Select the action to perform: • Run Component Method • Check File • Print File Head • Print File Tail • Publish JMS Text Message • Run Application Client • Run Component Method • Run Database Command • Run System Command • Send Mail Message • Send JMS Text Message
After Failure Run (afterFailureRun)	If the current task fails, select another task to perform. You can choose any of the tasks defined in Table 3-1 on page 18 or a user-defined task.
After Success Run (afterSuccessRun)	Select another task to perform after the current task completes. You can choose any of the tasks defined in Table 3-1 on page 18 or a user-defined task.
Thread Monitor (threadMonitor)	Select the thread monitor to use to perform the task.
Wait After Failure (waitAfterFailure)	If a task fails, the number of seconds to wait before allowing the task to run again.
Wait After Success (waitAfterSuccess)	If a task succeeds, the number of seconds to wait before allowing the task to run again.
Log Execution (logExecution)	Select to log scheduled task execution details.
Random Wait Offset (randomWaitOffset)	Select to randomize the intervals between running the task, to introduce variability.

Table 3-3 defines optional properties that depend on the value of taskAction. Ant property names are provided in parentheses after the Management Console display name. These options are appropriate for predefined and user-defined tasks.

Table 3-3: Type-specific options

Option	Description
Component Method (componentMethod)	If taskAction is runComponentMethod, specify the component method to run. The method must be void with no parameters. You can specify either: • The name method in a simple Java class (which must be in the class path); for example, com.example.MyTask.run, or • The name of a deployed component method; for example, ejb.components.mypackage.MyCompLocal.myMethod or ejb.components.mypackage.MyCompRemote.myMethod
Check File (checkFile)	If the task action is checkFile, specify which file to check.
Line Match (lineMatch)	Enter the character string to search for in the file.
Match if Count Exceeds (matchIfCountExceeds)	Specify the number of lines in which the pattern must be found to constitute a match.
Match if Delta Exceeds (matchIfDeltaExceeds)	Specify the number of new lines—added since the last check—in which the pattern must be found to constitute a match.
After Match Run (afterMatchRun)	If a line match is found, select another task to perform. You can choose any of the tasks defined in Table 3-1 on page 18.
Wait After Match (waitAfterMatch)	Specify the number of seconds to wait after a matching line is found before allowing the task to run again.
Print File Tail (printFileTail)	Specify the name of the log file.
Maximum Tail Lines (maximumTailLines)	Enter the maximum number of lines to print from the end of the file.
Mail Session (mailSession)	Specify the name of the mail session to use.
From (mailFrom)	Enter the sender’s e-mail address.
To (mailTo)	Enter the recipients’s e-mail address.
Cc (mailCc)	To send a carbon copy of a mail message, enter the recipient’s e-mail address.
Subject (mailSubject)	Enter the subject of the mail.
Message (mailMessage)	Enter the mail message body.
Append Output From (appendOutputFrom)	To append the output from a task as text in the mail message, select the task from those defined in Table 3-1 on page 18.
Attach Output From (attachOutputFrom)	To include the output from a task as an attachment in the mail message, select the task from those defined in Table 3-1 on page 18.

Option	Description
Print File Head (printFileHead)	Specify the name of the log file.
Maximum Head Lines (maximumHeadLines)	Enter the maximum number of lines to print from the beginning of the file.

Task properties on the Schedule tab

Properties on the Schedule tab define when to run the task. Table 3-4 describes the properties. Ant property names are provided in parentheses after the Management Console display name.

Table 3-4: Task schedules

Property	Description
Schedule (schedule)	Select the frequency at which to perform the task: • At specified interval • Every second • Every minute • Hourly • Daily • Weekly • Monthly • At server startup • At server shutdown • None
Start Time (startTime)	Specify the first time to perform the task, as `HH:MM`.
End Time (endTime)	Specify the last time to perform the task, as `HH:MM`.
Start Date (startDate)	Specify the first date to perform the task, as `YYYY-MM-DD`.
End Date (endDate)	Specify the last date to perform the task, as `YYYY-MM-DD`.
Exclude Date (excludeDate)	List specific dates to not perform the task, as `YYYY-MM-DD<, YYYY-MM-DD, ...>`.
Include Date (includeDate)	List specific dates to perform the task, as `YYYY-MM-DD<, YYYY-MM-DD, ...>`.

Property	**Description**
Day of Week (dayOfWeek)	To perform the task on specific days of the week, choose Select, then select or unselect each day. To perform the task every day of the week, select All. To create a schedule that is unrelated to days of the week, select None.
Day of Month (dayOfMonth)	To perform the task on specific days of the month, choose Select, then select or unselect each day. To perform the task every day of the month, select All. To create a schedule that is unrelated to days of the month, select None.
Month of Year (monthOfYear)	To schedule the task for specific months of the year, choose Select, then select or unselect each month. To perform the task every month of the year, select All. To create a schedule that is unrelated to months of the year, select None.

Ant configuration example

Below is an example Ant configuration script to define a scheduled task. For additional examples, see the file *default-scheduled-tasks.xml* in the *config* subdirectory of your installation. This configuration file defines the preconfigured scheduled tasks described in “Predefined tasks” on page 18.

Here is the example configuration script:

```
<?xml version="1.0"?>
<project name="default-scheduled-tasks" default="configure">
  <import file="ant-config-tasks.xml"/>
  <target name="configure">
    <setProperties scheduledTask="MyScheduledTask">
      <property name="taskAction" value="runComponentMethod"/>
      <property name="componentMethod"
value="ejb.components.myjar.MyCompLocal.myTask"/>
      <property name="schedule" value="interval"/>
      <property name="interval" value="30"/>
    </setProperties>
  </target>
</project>
```

Configuring scheduled tasks to run

Scheduled tasks defined in the repository do not run by default. You must install them into the server to run them. You can install a task in a server using the Management Console or by running an Ant configuration file.

To install using the Management Console, configure the Tasks tab in the Management Console Server Properties pages. For more information, see Chapter 3, “Creating and Configuring Servers,” in the *System Administration Guide*.

To install a scheduled task from an Ant configuration file, add it to the list of tasks in the server's scheduledTasks Ant configuration property. For example, if the task is “MyTask” and the server name is “MyApplicationServer”, add the following commands to your configure target:

```
<setProperties applicationServer="MyApplicationServer" merge="true">
  <property name="scheduledTasks"
            operation="append-list"
            value="MyTask"/>
</setProperties>
```

CHAPTER 4

Creating Service Components

This chapter describes how to create service components. Service components are loaded and initialized when EAServer starts and have a run method that executes perpetually, independent of any client interaction.

You can use service components to perform background processing or to provide common services for EAServer clients and other EAServer components.

Introduction

Service components perform background processing or provide common services for EAServer clients and other EAServer components. For example, you might create service components to perform the following tasks:

- Maintain cached copies of commonly used database tables
- Move or replicate data between data sources during server idle time
- Manage application-specific log files

What are service components?

Service components are like any other EAServer component, except that:

- They must implement the methods in the CtsServices::GenericService IDL interface. These methods allow the server to control the service component's initialization, execution, and shutdown.
- Instances are loaded and initialized when the host server starts.
- They can run independently of client interaction.

The Thread Manager and service components
You can use the Thread Manager as an alternative to creating a service component to handle repetitive processing. You may find the Thread Manager interface allows more design flexibility. For example, you can suspend processing in services run by the Thread Manager, and you can start threads at any time rather than only at server start-up. Chapter 5, "Using the Thread Manager" describes how to use the thread manager.

PowerBuilder developers can use the Thread Manager to develop more robust services. Since PowerBuilder components cannot support sharing and concurrency, you cannot develop a service that can be stopped or refreshed without using the Thread Manager. For more information, see the *Application Techniques* manual in the PowerBuilder documentation.

The GenericService interface

Your component's remote or local interface can include the methods in the CtsServices::GenericService IDL interface. EAServer calls these methods to control the service component's initialization, execution, and shutdown. Your implementation does not need to explicitly implement the interface (that is, list it in the implements clause of the class declaration), and you do not need to list the interface in the component properties. The methods are:

- **start()** Called to initialize the component when the server starts. This method typically initializes data structures and resources that the service requires. For example:
  - A service that writes to log files would open each file and cache each file handle as a class instance variable.
  - A service that caches tabular data from a remote database would open a connection to the database and create the data structures required to store tabular data in memory.
- **run()** Called after the first invocation of start() returns. run() can loop and perform repetitive tasks as an EAServer background process. If the component does not perform background processing, run() can return immediately.

  For services that perform background processing, run() should loop continuously while performing the service task. run() must periodically suspend its own execution by calling the Java java.lang.Thread.sleep() method, one of the Java Object.wait() methods, or the EAServer JagSleep C routine. These APIs suspend the current thread for a specified duration so that other threads may execute. run() should return after the server invokes the stop() method.

If you configure your service to run in multiple threads, EAServer calls run() concurrently in the specified number of threads.

Warning! Your run() method must either return immediately or call one of the Object.wait() Java methods, the EAServer JagSleep C routine, or some other thread-aware implementation of sleep. Do not call the sleep system routine or any other routine that suspends process (and not thread) execution. If coding service components in PowerBuilder, code your component to call the JagSleep C routine; do not use the PowerBuilder timer event, which may suspend the EAServer process.

- **stop()** Called when the server is shutting down or when the component has been refreshed (refresh stops the service then restarts the service cycle). EAServer calls the stop() method on a different thread than the run() method. Code in the stop() method should set a flag that indicates the the run() method should return.

  stop() should also wake up sleeping run() threads if the language allows this. For example, in Java, call the Object.notifyAll() method to wake threads that called Object.wait() on the same monitor object. In languages that do not allow you to wake up sleeping threads, keep your sleep interval reasonably short. The service cannot be refreshed until all running threads return from the run() method; that is, if your sleep interval is one hour, it can take that long to refresh the service unless you add code to wake up sleeping threads.

Implementing addtional remtoe interface methods

Your component can implement additional remote or local interface methods. EAServer clients, servlets, and other components can execute a service component's methods like those of any other component, with one exception: Clients cannot invoke methods on the service component until the start() method has returned. This restriction allows you to perform required initialization in start() without worrying about thread synchronization issues.

After start() returns, EAServer calls the run() method in its own thread. Client method invocations may arrive at this time as well. There is no guarantee that run() will have been called when a client method invocation occurs; the first client invocations may arrive before EAServer calls the run() method.

Creating service components

Follow the steps below to create a service component:

1. Create the service implementation
2. Implement GenericService interface methods
3. Create a service component entity
4. Install the service component in the server

Create the service implementation

Service components are implemented using EJB stateless session beans or stateless CORBA or PowerBuilder components. Except for a few special requirements described here, you define a service component's interface and properties as you would do for any component.

Component properties

The EJB or CORBA component that implements your service requires these settings:

- **Remote or local interface** The component's remote or local interface must include the methods in the CtsServices::GenericService IDL interface. You can define additional methods if necessary.
- **Concurrency and threading options** For best performance, your component must be thread-safe and allow concurrent execution. To support concurrency, you must ensure that access to read/write instance variables is synchronized in your component.

  If using CORBA components, enable the Thread Safe property, and disable the Bind Thread option.

  Enabling the Thread Safe property allows multiple method invocations to occur simultaneously. If your component has a run() method that executes indefinitely, you must enable the Concurrency option or no clients will be able to invoke methods. To stop the service, EAServer must invoke the stop method on a thread executing concurrently to the thread executing the run method.

  Disabling the Bind Thread option allows EAServer to run the component on any available thread. This option is only required by components that use thread-local storage. It should be disabled in any other case.

- **Transaction attribute** Do not create service components that are transactional. EAServer-managed transactions require a component lifecycle that allows component deactivation, and a service component is never deactivated. The component's transaction attribute must be Not Supported. If you require EAServer's transaction semantics, implement a component to perform the transaction-created work and call this component from your service component.

Required client roles

You can assign the role ServiceControl to service components so that base clients and other components cannot create instances of the component and call the start and stop methods. No users can be added to this role.

Implement GenericService interface methods

Each service component must implement the CtsServices::GenericService interface methods. This section describes how to implement the CtsServices::GenericService in C++ and Java.

Be careful of consuming CPU cycles
If your service will perform background processing, your implementation must have access to a thread-aware sleep mechanism. In Java, call the java.lang.Thread.sleep() method, or use a monitor object and call the Object.wait() method. In C, C++, or PowerBuilder, EAServer provides the JagSleep routine. The run method in your service must call one of these APIs periodically to suspend execution of the current thread. Otherwise, your service will dominate the server's CPU time and prevent other components from executing.

If coding service components in PowerBuilder, code your component's run method to call the JagSleep C routine; do not use the PowerBuilder timer event, which may suspend the EAServer process.

Services with a client interface
If your component runs as a service and also provides a client interface for remote invocations, beware that the run method may not have executed when the first client request arrives. run is called on a different thread after start returns; client invocations may arrive between the return from start and the invocation of run, and initialization performed in run may not have completed when the remote method executes on a different thread. To avoid problems, use one of these approaches:

- Do not code remote methods that rely on initialization performed in the run method. Initialization can be performed in the start method, which is guaranteed to complete before client invocations arrive.
- Use a synchronized boolean variable that is set when run has performed necessary initialization, and code remote methods to check this variable and wait for it to be set before executing code that relies on initialization performed in run.

Java example of GenericService methods

The example uses a static Boolean instance variable, *_run*, to indicate when the service should cease running. There is also a java.lang.Object that is used as a semaphore to allow synchronization among multiple threads. The start() method sets the *_run* variable to true; start() must also perform any other necessary initialization that are needed by your service, such as opening files, database connections, and so forth. run() executes a while loop as long as the *_run* variable is true. In each loop iteration, run() performs some of the work that the service is responsible for, such as refreshing a copy of a remote database table, then calls the Object.wait() method to relinquish the CPU. The stop() method sets the *_run* variable to false and calls the Object.notifyAll() method on the semaphore, causing the run() method to return. Before returning, run() cleans up resources that were allocated in the start() method.

```
public class MyService
{
public static boolean _run;
public static Object _lock = new Object();

public void start()
{
    _run = true;
    ... perform necessary initializations ...
}
public void run()
{
    while (_run)
    {
        try
        {
            ... do whatever this service does
                on each iteration, then go back
                to sleep for a while ...
            synchronized(_lock)
            {
                _lock.wait(100000);
            }
        }
        catch (InterruptedException ie)
        {
            _run = false;
        }
    }
    ... perform necessary cleanup and deallocations ...

}

public void stop()
{
    _run = false;
    // Wake up any instances that are waiting on the mutex
    synchronized (_lock)
    {
        _lock.notifyAll();
    }
}
}
```

C++ example of GenericService methods

The code fragment below shows how the GenericService methods can be implemented in a C++ component. This example uses a static Boolean instance variable, *_stop*, to indicate when the service should cease running. The start() method sets the *_stop* variable to false; start() must also perform any other necessary initialization that are needed by your service, such as opening files, database connections, and so forth. run() executes a while loop as long as the *_stop* variable is false. In each loop iteration, run() performs some of the work that the service is responsible for, such as refreshing a copy of a remote database table, then calls the JagSleep C routine to relinquish the CPU. The stop() method sets the *_stop* variable to true. stop() must also clean up any resources that were allocated in the start() method.

```
#include <jagpublic.c> // For JagSleep API

class MyService
{
private:
    static boolean _stop; // Declared static in case multiple
                          // instances are run.

public:
void start()
{
    _stop = false;
    ... perform necessary initializations ...
}

void stop()
{
    _stop = true;
}

void run()
{
    while (! _stop)
    {
        ... do whatever this service does
            on each iteration ...
        JagSleep(1000);
    }
    ... perform necessary cleanup and deallocations ...
}

};
```

Create a service component entity

Before you can install the service component in a server, you must create a service component entity using the Management Console or a configuration script.

❖ **Creating a service component entity with the Management Console**

1. Highlight the Services Components folder and click Add. The New Service Component wizard launches. Specify the service component name.
2. Configure the properties listed in Table 4-1 on page 36.

❖ **Creating a service component entity with a configuration script**

1. Create a configuration script like the example below.
2. Run the configure target to define the service component.

Example service component configuration script This Ant script creates a service component named MyService and installs it in the default server configuration. For details on the properties, see Table 4-1 on page 36.

```
<?xml version="1.0"?>
<project name="create-service-comp" default="configure">
  <import file="ant-config-tasks.xml"/>
  <target name="configure">
<setProperties serviceComponent="MyService" merge="false">

        <property name="component" value="ejb.components.my_ejb_service.MyEJB
ServiceRemote"/>
        <property name="dependsOn" value="JaguarServer"/>
        <property name="startOrder" value="2"/>
        <property name="startBeforeBinding" value="true"/>
        <property name="runThreadCount" value="2"/>
        <property name="stopWaitTime" value="1000"/>
        <property name="allowStartFailure" value="true"/>
    </setProperties>

    <setProperties applicationServer="default" merge="true">
        <property name="serviceComponents" operation="append-
value" value="MyService" />
    </setProperties>  </target>
</project>
```

Service component entity properties

Table 4-1 lists the service component entity properties.

Table 4-1: Service component entity properties

Management Console property	Configuration file property	Description
Component	component	Identifies the component to run as the service. Specify the identifier for the component's generated DJC adapter for the EJB remote or local interface. For remote interfaces, this has the form: ejb.components.*module*.*component*Remote Where *module* is the name of the EJB module or CORBA package, and *component* is the name of the component. For local interfaces, it has the form: ejb.components.*module*.*component*Local For example, for component MyService in module MyPackage, specify the remote interface adaptor as `ejb.components.MyPackage.MyComponentRemote`.
Depends On	dependsOn	Specifies other service components that must be running before this component can run.
Start Order	startOrder	Allows you to configure the starting order for multiple services. Lower numbered services start before higher number services.
Start Before Binding	startBeforeBinding	Specifies whether EAServer must perform JNDI name bindings before calling the start method. Disable this setting if your start method makes intercomponent calls or accesses resources using JNDI.
Run Thread Count	runThreadCount	Specifies how many instances to run. Each instance runs in a separate thread—see "When multiple threads are requested" on page 36
Stop Wait Time	stopWaitTime	When stopping the service, how long to wait for the stop method to return before giving up. Specify the time in milliseconds.
Allow Start Failure	allowStartFailure	Whether the server can run and accept client connections if this service fails to start.

When multiple threads are requested

The host server calls the component's run method from the specified number of threads. If the Sharing option is enabled, all threads call run on the same component instance as start was called in. Otherwise, each thread will create a new instance of the component and call run on that instance. Each thread terminates when run returns.

This feature is useful when your service component performs a background task that lends itself to parallel processing. For example, if the run implementation extracts work requests from a queue and performs the requested operation, you can configure the server so multiple threads read requests from the queue and process them simultaneously. The component must be coded to ensure that access to the queue is thread-safe, for example, in Java, you might create synchronized methods to queue and dequeue.

The component must be stateless in order to run in multiple threads.

Note The start method and stop methods are only called on one instance of a service component. If Sharing is not enabled for the component, start must store any data required by the run method or other methods in a way that allows access from other class instances. For example, use static class fields or a persistent data store.

Install the service component in the server

In order to run as a service, your service component must be added to the host server's list of services, as follows:

❖ **Installing services**

1. Click the icon for the server in the Servers folder. To install a service as part of the default server configuration, click the default server icon.

   The server properties display.

2. Display the Services tab.

3. Available service components are displayed with a check box for those that are installed. Install your service by selecting the corresponding check box.

   Click Apply to save your changes.

4. The service runs the next time you refresh or restart the server.

Determining service state

The jagtool getservicestate command returns the state of service components executing in the server. You must code your service component to implement the methods of the CtsServices::ExtendedService IDL interface to allow users to query the component state with jagtool.

This interface extends CtsServices::GenericServices, and adds one method:

```
long getServiceState()
```

This method must return one of the constants listed in Table 4-2 to describe the state of the service. These constants are defined in module CtsServices.

Table 4-2: Service states

State	Description
`UNKNOWN`	The state is unknown.
`STARTING`	The service is starting. The start method has been called, but has not returned.
`STARTED`	The service is started, but not yet running. The start method has returned, but run has not been called.
`RUNNING`	The service is running, that is, executing the run method.
`FINISHED`	The service is finished processing. The run method has returned. This state applies only to services that do not run continuously until stopped.
`STOPPING`	The service is stopping. The stop method has been called, and is still running.
`STOPPED`	The service is stopped. The stop method has been called and has returned.

The following Java example shows service component code that determines and returns state:

```
import CtsServices.*;

...

public class MyService
{
    private static boolean _starting = false;
    private static boolean _running = false;
    private static boolean _stopping = false;
    private static boolean _stop = false;
    private static boolean _runHasBeenCalled = false;
    private static Object _lock = new Object();
    public void start()
```

```
{
    _starting = true;
    // Perform initialization
    _starting = false;
}
public void stop()
{
    _stopping = true;
    _running = false;
    _stop = true;
    synchronized (_lock)
    {
        _lock.notifyAll();
    }
    // Perform cleanup
    _stopping = false;
}
public void run()
{
    _runHasBeenCalled = true;
    // Perform per-thread initialization here.
    _running = true;
    while (! _stop)
    {
        try
        {
            // do whatever this service does on
            // each iteration
            synchronized(_lock)
            {
                _lock.wait(100000);
            }
        }
        catch (InterruptedException ie)
        {
            _stop = true;
        }
    }
    // Perform per-thread cleanup here.
    _running = false;
}
public int getServiceState()
{
    if (_starting)
    {
        return SERVICE_STATE_STARTING.value;
```

```
            }
            else if (! _runHasBeenCalled)
            {
                return SERVICE_STATE_STARTED.value;
            }
            else if (_stopping)
            {
                return SERVICE_STATE_STOPPING.value;
            }
            else if (_stop)
            {
                return SERVICE_STATE_STOPPED.value;
            }
            else if (_running)
            {
                return SERVICE_STATE_RUNNING.value;
            }
            else
            {
                return SERVICE_STATE_FINISHED.value;
            }
        }
    }
```

Refreshing service components

Service components are not refreshed when you refresh the package or server in which the component implementation is installed. You must explicitly refresh the service component entity. To refresh a service component in the Management Console, right-click the icon for the service component entity and choose Refresh.

EAServer refreshes a service component as follows:

1 The server calls the stop() method. Your implementation must communicate the stoppage so that the run() method returns in each thread running the service.

2 The server waits for the run() method to return in all instances that are running as services. The wait time can be configured with the service component Stop Wait Time property.

3. The server creates a new instance and calls the start() and run() methods, in that order. If the multiple instances are specified for the service, the server loads the additional instances that are required and calls run() on each instance.

After refresh, a new instance is guaranteed not to start before previous instances have ceased running. Consequently, a service component can not be refreshed unless the run() method returns. See "Implement GenericService interface methods" on page 31 for code examples that show how to coordinate the logic in the stop() and run() methods.

CHAPTER 5 Using the Thread Manager

The Thread Manager allows you to start threads from EAServer components to perform asynchronous processing.

About the Thread Manager

The Thread Manager allows you to run EAServer component instances in threads that execute independently of client method invocations. You can use threads spawned by the Thread Manager to perform any processing that must occur asynchronously with respect to user interaction. For example, you might have a component method that begins a lengthy file indexing operation. The method could call the Thread Manager to start the processing in a new thread, then return immediately.

The Thread Manager and service components

For performing repetive background task processing, the Thread Manager provides an alternative and complementary technology to service components or scheduled tasks. You may find the Thread Manager interface allows more design flexibility. For example, you can suspend processing in services run by the Thread Manager, and you can start threads at any time rather than only at server start-up.

The Thread Manager is the recommended way to spawn threads in Java or C++ components. In C++, using the Thread Manager avoids system-level thread calls that may affect portability. In Java and C++, components running in the Thread Manager can make in-memory intercomponent calls, whereas components running in user-spawned threads must make intercomponent calls through the network.

You can use the Thread Manager and service components together. For example, you might code a simple service component that spawns threads in the start or run method, and stops them in the stop method.

PowerBuilder developers can use the Thread Manager to develop more robust services. Since PowerBuilder components cannot support sharing and concurrency, you cannot develop a service that can be stopped or refreshed without using the Thread Manager. In the services start or run method, spawn threads that do the service's processing. In the service's stop method, call the Thread Manager stop method to halt the threads. For more information, see the *Application Techniques* manual in the PowerBuilder documentation.

The Thread Manager and the message service

If you are using threads to implement a provider/consumer algorithm, or an asynchronous notification system, consider using the EAServer message service implementation described in the *Java Message Service User's Guide*. The message service provides a ready-made infrastructure for solving these classes of problems.

The Thread Manager and scheduled tasks

Using the scheduled tasks feature, you can restrict background processing to a server's off-peak hours. For example, you may have threads running that index the text content of a Web site. Using a scheduled task and the Thread Manager, you can suspend processing at the beginning of the server's peak use period, then resume processing at the end. For details on configuring scheduled tasks, see Chapter 3, "Using Scheduled Tasks."

Thread Manager interface documentation

This chapter briefly discusses how to use the Thread Manager methods. For reference documentation, see the generated HTML documentation for the CtsComponents::ThreadManager IDL interface. You can view this documentation in the *html/ir* subdirectory of your EAServer installation. Using a Web browser, load the URL:

```
http://host:port/ir/index.html
```

Where *host* is your server's host name, and *port* is the HTTP port number.

Using the Thread Manager

The Thread Manager is a built-in EAServer component. You can create a proxy and execute methods the same way that you would call any other component. Each thread executes a run method in an EAServer component that you specify.

The thread manager is designed primarily for use in server-side code. However, it is possible to call thread manager methods from base clients or Web applications. For example, you can create an administrative client that stops threads created by your application.

Before you start

Before running components in the Thread Manager, make sure you understand how the component must be prepared, how threads are run in thread groups, and the effect of a thread group's run interval.

Adapting components to be run by the Thread Manager

Each thread runs an EAServer component instance. To be run by the Thread Manager, the component's remote or local interface must have a run method that matches this IDL signature:

```
void run ( );
```

The Thread Manager calls the run method one or more times, depending on how you configure the run interval (described below).

The Thread Manager is itself an EAServer component, and runs your component using intercomponent calls. All component properties, including transaction attributes, are in effect when your component is run by the Thread Manager. The Thread Manager executes with the system identity, as does your component's run method.

Understanding thread groups

Threads are associated with a thread group. To start, stop, suspend, or manage the run interval of threads, you must specify the group name. These operations affect all threads in the specified group. The group name is simply a string. Group names have a scope limited to one server; that is, you cannot have two like-named groups in the same server. If two applications use the same group name, their Thread Manager calls affect threads in both applications. You can run different components in one thread group.

Naming conventions for thread groups
To avoid collisions between thread groups used by different applications, use the reverse-domain naming convention for group names, as used in Java package names. For example, “com.foo.mythreadgroup”.

Understanding the run interval

Each thread group has a run interval, which determines how often the Thread Manager calls the run method. The run interval can be:

Run interval	Meaning
A positive integer *n*	The Thread Manager calls run repeatedly, waiting approximately *n* seconds after each time the run method returns. The actual time can vary depending on scheduling of calls to other methods and the server’s processing load.
0	The Thread Manager calls run repeatedly, with no waiting between invocations.
-1 (the default)	The Thread Manager calls run only once.

To allow threads to be stopped or suspended, you must configure a positive or 0-length run interval and code each component’s run method to perform a repetitive task, then return. The run interval has no effect if your run method never returns.

If the run interval is positive or 0, you can change the run interval after threads have been started in the group, the change takes for each thread when it returns from the run method. You cannot change the interval to -1, and changing the interval does not affect threads started with the interval set to -1. In these cases, calling setRunInterval has no effect.

You can use a run interval to schedule periodic tasks, such as refreshing a cached copy of a database query result. You can also tune how much CPU time your component consumes if it performs CPU-intensive tasks such as lengthy calculations; such tuning also requires that you adjust the amount of work done in each invocation of the run method.

You can also use the scheduled tasks feature to perform background processing. For details, see Chapter 3, “Using Scheduled Tasks.”

Understanding the thread count

Each thread group has a thread count, which determines how many threads can run simultaneously. The count can be:

Run interval	Meaning
-1 (the default)	There is no limit.
A positive integer *n*	*n* threads can execute.
0	No threads can execute.

To change the thread count, call the ThreadManager::setThreadCount method. The change takes effect after threads return from the run method. Thread counts are useful if threads run repeatedly (run interval is positive or 0). For example, if 6 threads are running, and you change the count to 5, the next thread that returns from its run method will not be restarted. The thread count provides a means to throttle the number of running threads, without stopping or suspending all threads.

Instantiating the Thread Manager

Other than restricted access, the Thread Manager can be instantiated as you would instantiate any other component.

Obtaining authorized access

To instantiate the Thread Manager, your client or component must execute with with the system identity or an identity that is in the ThreadManager role. These are the recommended ways to satisfy this constraint:

- Start threads from a service component and create the Thread Manager proxy in the service's start or run method. These methods execute with the system identity.
- For a component that is pooled or shared, create the Thread Manager proxy in the component's class constructor, the setSessionContext or setEntityContext method (for EJB components), or the setObjectContext method (for CORBA components). All of these methods execute with the system identity.
- For a component that is not a service and not pooled or shared:
    - Delegate Thread Manager operations to another component that is pooled or shared, or

    - Run the component with an identity that is in the ThreadManager role.

- For a base client, connect to EAServer with a user name that is a member of the ThreadManager role.

ThreadManager privileges can be dangerous
User accounts with ThreadManager role membership can use the Thread Manager to implement denial of service attacks or to stop thread groups. Treat ThreadManager role accounts with the same care as you would Admin role accounts.

Instantiating a proxy

Use the standard technique for your component model to instantiate the Thread Manager proxy.

CORBA (C++ and Java), and PowerBuilder components must declare a stub for the CtsComponents::ThreadManager IDL interface, then instantiate the component named *CtsComponents/ThreadManager*.

EJB components must use the home interface com.sybase.ejb.cts.ThreadManagerHome to create a stub for the remote interface com.sybase.ejb.cts.ThreadManager. Look up the name *CtsComponents/ThreadManager* to obtain the home interface.

Starting threads

To start threads:

1. Optionally, configure a run interval by calling the setRunInterval method, specifying the group name.
2. If necessary, create proxies for the components that will run in the thread group. For stateless or shared-instance components, you can use one proxy instance to run the component on multiple threads. For stateful components, create a proxy for each component instance and initialize the instance state as necessary.
3. Start the desired number of threads by calling the start method once per thread. In each call, specify the group name and pass a proxy for the component that is to run in the thread.

If you have set a thread count, and try to start more threads than the thread count, the behavior depends on the run interval. If the run interval is -1, all threads are started and run once. If the run interval is 0 or positive, the start method does not create additional threads after the count is reached.

Suspending and resuming execution

To suspend the threads in a group, call the ThreadManager::suspend method, specifying the group name. Each thread is suspended when it next returns from its run method.

To resume execution, call the ThreadManager:resume method.

Stopping threads

You can only stop threads that return from their run method. The Thread Manager stops each thread the next time it returns from its run method.

You can stop threads in two ways:

- **By decreasing the thread count** Call the ThreadManager::setThreadCount method to reduce the number of threads executing in the thread group. This technique is useful when you want to throttle the execution of the task. For example, during a Web site’s peak usage hours, you can reduce the thread count for background processing to give user threads more CPU time. During off hours, you can reset the thread count and start new threads to raise the thread count again.
- **By stopping all threads in the group** Call the ThreadManager::stop method to stop all threads in the group. This method is equivalent to calling ThreadManager::setThreadCount to reduce the thread count to zero.

If you stop all threads by calling ThreadManager::stop or setting the thread count to 0, you must reset the thread count to a positive value or -1 (meaning infinity) before starting more threads.

CHAPTER 6 Using jagtool and jagant

jagtool is a command line interface that allows you to automate some EAServer development and deployment tasks. You can use jagtool from the command line, from scripts or makefiles, or with Jakarta Ant.

This chapter contains instructions on how to use jagtool, either by itself, or with jagant.

Beginning with EAServer 6.0, jagtool and jagant are supported for backward compatibility. Sybase recommends that for new development you use Ant configuration scripts, as described in Chapter 2, "Ant-Based Configuration," and command line tools, such as wsh, wfs, and deploy documented in Chapter 12, "Command Line Tools," in the *System Administration Guide*.

Working with jagtool

Before using jagtool, make sure that:

- The DJC_HOME environment variable is set.
- **UNIX** *$DJC_HOME/bin* is added to your path.
- **Windows** *%DJC_HOME%\bin* is added to your path.

Use the following scripts to run jagtool:

- **UNIX** *$DJC_HOME/bin/jagtool*
- **Windows** *%DJC_HOME%\bin\jagtool.bat*

jagtool syntax

The syntax for jagtool is:

jagtool [*connect-args* | *local-args*] [*log-arg*] [*command*]

Where:

- *connect-args* is a list of arguments required to run in connected mode.
- *local-args* is a list of arguments required to run in local mode.
- *log-arg* is an optional argument to specify a file name to record jagtool output. If you do not specify a file name, output is sent to the standard output device. Specify a file name using the `-logfile` ***`filename`*** argument or `-l` ***`filename`***.
- *command* is a jagtool command described in “jagtool commands” on page 66.

Local versus connected mode

You can run jagtool in either connected mode or in local mode. In connected mode, jagtool connects to a server that can be running locally or on a remote machine. In local mode, jagtool does not require a connection to a server, but you can run it only on a machine that has file system access to the EAServer installation.

Using connected mode

Connected mode is jagtool’s default mode of operation. All commands can run in connected mode. When using connected mode, specify the arguments listed in Table 6-1.

Table 6-1: jagtool connection arguments

Parameter	To specify
`-h` *`hostname`* or `-host` *`hostname`*	Server host name. If not specified, the default is the value of the HOSTNAME environment variable.
`-n` *`port`* or `-port` *`port`*	Server IIOP port number. If not specified, the default is 9000.
`-u` *`name`* or `-user` *`name`*	User name. If not specified, the default is "admin@system".
`-p` *`password`* or `-password` *`password`*	Password. If not specified, the default is "" (no password).

For example, to connect to the server running on "eclipse" at port 9005, using account admin@system with password "secret" enter:

```
jagtool -h eclipse -n 9005 -p secret
```

You can omit the *-u* flag because *admin@system* is the default user name.

Unless otherwise specified in the command reference page, all commands can run in connected mode.

Using local mode

Local mode allows you to configure an EAServer installation without requiring a connection to a server. Local mode is helpful in situations where it is not convenient to start a server, for example, if you are using jagtool to configure new installations. You cannot run all commands in local mode. To see whether a command supports local mode, check the syntax listing for the command in this chapter or check the help output for the command.

To run in local mode, specify the arguments in Table 6-2.

Table 6-2: jagtool local-mode arguments

Parameter	To specify
`-local`	Specifies whether to run in local mode, without a connection to the server. If you do not specify this parameter, jagtool requires a connection to a running server.
`-server` *`servername`*	Specifies the name of the server to use when running in local mode. Specify the name of the server that you would connect to if running in connected mode. If you do not specify a server name, the default is the name of the machine on which EAServer is installed.

Local mode also requires the following:

- You must be logged in to the operating system as a user with read and write privileges on the EAServer installation directory and all subdirectories.
- You must set the DJC_HOME environment variable to specify the local installation directory.

When running jagtool or jagant locally, the user name and password arguments are ignored.

Entity identifiers

Many jagtool commands take one or more entity identifiers as arguments. An entity identifier is a string of the form *EntityType:EntityName* that uniquely identifies an entity in the repository. *EntityName* is the J2EE module name; that is, the name of the archive from which the entity is deployed. For example, if you deploy *myejb.jar*, the entity name is “myejb.”

Table 6-3 provides examples of entity identifiers for each entity type.

Table 6-3: Example entity identifiers

Entity identifier	Specifies
`Agent:agent1`	Agent named agent1.
`Application:estore`	Application named estore.
`ApplicationClient:estore/PetStoreClient`	Application client named PetstoreClient in application named estore.

Entity identifier	Specifies
`Cluster:TheBigCluster`	Cluster named TheBigCluster.
`Component:SVU/SVULogin`	Component named SVULogin that is installed in the SVU package. The package name is included because EAServer components always reside in packages.
`ConnCache:JavaCache`	Connection cache named JavaCache.
`Connector:BlackBoxLocalTx`	J2EE connector named BlackBoxLocalTx.
`DatabaseType:Sybase_ASA`	Database type definition named Sybase_ASA.
`EntityCollection:MyEntityCollection`	Entity collection named MyEntityCollection.
`Filter:WebTier/MyFilter`	Servlet filter named MyFilter installed in the Web application named WebTier.
`InstancePool:MyPool`	The named instance pool MyPool.
`Listener:Jaguar/iiops1`	The network listener named iiops1 installed in the server named Jaguar. When specifying a listener name, use the server name and the listener name as displayed in the Management Console.
`ManagedConnectionFactory:BlackBoxLocalTx/EASDemo`	The managed connection factory named EASDemo in the J2EE connector named BlackBoxLocalTx.
`Method:SVU/SVULogin/isLogin`	Method named isLogin of component SVULogin in package SVU.
`Package:SVU`	Package named SVU.
`Role:MyRole`	Role named MyRole.
`Security:sample1`	Security entity named sample1.
`Server:Jaguar`	Server named Jaguar.
`Service:MyPack/MyComp`	Service component named MyComp, installed in package MyPack. Use this syntax to install or remove service components from a server.
`Servlet:StandAloneServlet`	The servlet named StandaloneServlet. This syntax is valid only for servlets that are not installed in a Web application.
`Servlet:MyWebApp/MyServlet`	Servlet named MyServlet in Web application named MyWebApp. You must use this syntax for servlets that are installed in a Web application.
`WebApplication:WebTier`	Web application named Web tier.

Not all jagtool commands support every type of entity in the repository. For example, the refresh command is not supported for the Listener entity type.

When a command specifies an invalid entity type, an appropriate error message is displayed.

jagtool and *jagant*

jagant lets you run jagtool commands from Ant build files. This powerful feature allows you to write build files that automate many development and deployment tasks.

Jakarta Ant is a Java-based build tool developed by the Apache Jakarta project. To obtain Ant software and documentation, see the Ant Web site at http://jakarta.apache.org/ant/. Ant works similarly to other build tools (such as *make*, *gnumake*, or *jam*) but is platform-independent, extending Java classes rather than OS-specific shell commands. Ant includes a number of tasks that are frequently used to perform builds, including compiling Java files and creating JAR files. It also includes common functions such as copy, delete, chmod, and so on.

Ant build files (similar to a *make* file) are written in XML. Like *make*, Ant build files can include targets that perform a series of tasks. Instead of extending shell commands, Ant's build file calls out a target tree where various tasks are executed. Each task is run by an object that implements a particular task interface.

Setting up your environment

Install Ant and read the accompanying documentation.

The jagant script requires a full JDK installation. If you are running jagant from an EAServer client install, make sure you have installed the full JDK. By default, only the JRE files are installed.

Before running jagant, verify that:

- The DJC_HOME environment variable is set.
- A full JDK installation is present.
- Jakarta Ant is installed on your system.

By default, jagant searches for Jakarta Ant in *%DJC_HOME%\ant* (Windows) or *$DJC_HOME/ant* (Solaris). To use a different Ant installation, set the ANT_HOME environment variable before running the jagant script to specify the Ant installation location.

- If you are using jagant to compile JSP files with the compilejsp command, modify the CLASSPATH setting for the Ant scripts, adding the location of the *xalan.jar* and *crimson.jar* files that are included with EAServer. For example, if you are using Windows, edit the *ant.bat* file, and change the code under the `:runAnt` label to read:

```
:runAnt
set DJC_HOME=EAServer installation directory
set LOCALCLASSPATH=%DJC_HOME%\java\classes\crimson.jar;%LOCALCLASSPATH%
set LOCALCLASSPATH=%DJC_HOME%\java\classes\xalan.jar;%LOCALCLASSPATH%
%_JAVACMD% -classpath %LOCALCLASSPATH% -Dant.home="%ANT_HOME%"
%ANT_OPTS% org.apache.tools.ant.Main %ANT_CMD_LINE_ARGS%
goto end
```

Or on UNIX, change the last line to read like these lines:

```
DJC_HOME=EAServer installation directory
LOCALCLASSPATH=$DJC_HOME/java/classes/crimson.jar:$LOCALCLASSPATH
LOCALCLASSPATH=$DJC_HOME/java/classes/xalan.jar:$LOCALCLASSPATH
$JAVACMD -classpath "$LOCALCLASSPATH" -Dant.home="${ANT_HOME}" $ANT_OPTS
org.apache.tools.ant.Main "$@"
```

jagant scripts

The following scripts are provided for running Ant with jagtool commands:

- **Windows** *%DJC_HOME%\bin\jagant.bat*
- **UNIX** *$DJC_HOME/bin/jagant.sh*

jagant syntax

The jagant script uses this syntax:

jagant *[ant_options]*

where *ant_options* are any options and commands supported by Ant; see the Ant documentation for details on these options.

You may frequently use the *-buildfile* flag. Using *-buildfile* lets you specify a location other than the default for the Ant XML build file.

As with jagtool, you can run jagant in local or connected mode. “Local versus connected mode” on page 52 explains the difference.

The Ant build file

To use jagant, you must create an Ant build file that imports the jagant task definitions, specifies a connect task to connect to EASever, and runs the intended tasks.

A sample build file

Here is a sample build file:

```
<?xml version="1.0"?>
<!DOCTYPE project [
    <!ENTITY jagtasks SYSTEM "file:./jagtasks.xml">
]>
<project name="sample" default="refresh_svu" basedir=".">

    <!-- include Jaguar task definitions -->
    &jagtasks;

    <!-- global properties for this build -->
    <property name="jaguar.host" value="SANDVIK2K1" />
    <property name="jaguar.port" value="2000" />
    <property name="jaguar.user" value="jagadmin" />
    <property name="jaguar.password" value="easerver6" />

    <!-- connect -->
    <target name="connect">
        <jag_connect host="${jaguar.host}" port="${jaguar.port}"
user="${jaguar.user}" password="${jaguar.password}" />
    </target>

    <!-- refresh package ejbtut -->
    <target name="refresh_ejbtut" depends="connect">
        <jag_refresh entity="Package:ejbtut" />
    </target>
    <!-- restart the server -->
    <target name="restart_server" depends="connect">
        <jag_restart />
    </target>

    <!-- shutdown the server -->
    <target name="shutdown_server" depends="connect">
        <jag_shutdown />
    </target>

</project>
```

This sample imports the EAServer *jagtasks.xml* file to declare the jagant tasks. The *jagtasks.xml* file is located in the EAServer *config* subdirectory. The syntax shown in the sample imports the file from the same directory; to run build files using this syntax, copy *jagtasks.xml* to the same directory as your build file, place a copy of your build file in the EAServer *config* subdirectory, or edit the path specified by the system entity declaration, for example:

```
    <!ENTITY jagtasks SYSTEM "file:../config/jagtasks.xml">
```

To run jagant with the sample build file, enter this command all on one line:

```
jagant -buildfile sample.xml refresh_ejbtut
```

In this example, jagant is invoked with the specified build file. The target, *refresh_ejbtut*, refreshes the package named *ejbtut* by invoking the jag_refresh command. You can run other targets in the sample using the same syntax.

Most targets in the sample depends on the connect target. This dependency ensures the connection is established when the target runs. The connect target invokes the jag_connect command to open a connection with the server. The host name, port number, user name and password are defined as Ant properties in the build file.

You can override these property values at the command line using the Ant `-D` option. This is typically done to specify the password, so that it is not stored directly in the build file. For example (entered all on one line):

```
jagant -Djaguar.host=eclipse -Djaguar.port=9005 -Djaguar.password=jagpass
-buildfile sample.xml refresh_ejbtut
```

This command connects to the server with a host name "eclipse" on port 9005, with the user name admin@system and the password "jagpass." The default user of admin@system is still used because it was not overridden at the command line.

Registering *jagtool* commands in the Ant build file

Each build file that invokes jagtool commands must include definitions for those commands. An Ant *taskdef* directive is required for each jagtool command. The EAServer *jagtasks.xml* file in the *config* subdirectory contains the necessary directives. Import this file into your build files using the syntax shown in "A sample build file" on page 58.

Using the *jag_connect* command

In build files, use the jag_connect command to connect to a server or to specify the server name for local mode. You cannot use jag_connect from the command line; instead use the connection or local-mode arguments described in "Local versus connected mode" on page 52.

jag_connect must be executed before any other jagtool commands in the build file. jag_connect can be included directly in any target, or in a "connect" target that other targets list as a dependency.

Using *jagant* in connected mode

To run jagant in connected mode, specify these options for the jag_connect command:

- **host** The name of the host where EAServer is running.
- **port** The port number for the server. The default is 9000.
- **user** The user name used to connect. The default is admin@system.
- **password** The password used to connect. The default is no password.
- **logfile** The log file for the connection attempt. The default is *System.out*.

For example, this sample project defines a connect task to connect to the machine “myhost” at port 9000, logging in as “admin@system,” and runs the jag_list command over the connection:

```
<?xml version="1.0"?>

<!DOCTYPE project [
<!ENTITY jagtasks SYSTEM "file:./jagtasks.xml">
]>

<project name="local_sample" default="list_packages" basedir=".">

  <!-- include server task definitions -->
  &jagtasks;

  <!-- connect -->
  <target name="connect">
    <jag_connect host="myhost" port="9000" user="admin@system" password="" />
    </target>

  <!-- list packages in the server -->
  <target name="list_packages" depends="connect">
    <jag_list type="Package" />
  </target>

  <!-- list the properties of package CtsSecurity -->
  <target name="CtsSecurity_props" depends="connect">
    <jag_props entity="Package:CtsSecurity" />
  </target>

</project>
```

Using *jagant* in local mode

To define a jag_connect task to run in local mode, set the localServer option to the name of the server to use when running in local mode. Specify the name of the server that you would connect to if running in connected mode. If you specify this option, the connected-mode arguments are ignored and jagant runs in local mode. For example, this sample project defines a connect task to run in local mode on the server *Jaguar*, then runs the jag_list command in the scope of the local-mode connection:

```
<?xml version="1.0"?>

<!DOCTYPE project [
<!ENTITY jagtasks SYSTEM "file:./jagtasks.xml">
]>

<project name="local_sample" default="list_packages" basedir=".">

    <!-- include server task definitions -->
    &jagtasks;

    <!-- connect -->
    <target name="connect">
        <jag_connect localServer="Jaguar" />
    </target>

    <!-- list packages in the server -->
    <target name="list_packages" depends="connect">
        <jag_list type="Package" />
    </target>

</project>
```

Using multiple connections

You can use multiple jag_connect commands in a single build file, which allows you to execute jagtool commands for different servers. Each time jag_connect is executed, the current connection or local-mode session is closed and a new one is opened. For example, this code restarts two servers:

```
<target name="restart_all_servers">
<jag_connect host="host1" password-="jagpass" />
<jag_restart />
<jag_connect host="host2" password="jagpass" />
<jag_restart /></target>
```

A connection to “host1” is opened and the server on “host1” is restarted. Then the connection is closed, a connection is opened to “host2,” and the server on “host2” is restarted. The port number for both servers is 9000 and the user name is admin@system (the defaults). The password for both servers is “jagpass,” and the log file is *System.out*.

XML configuration files

Rather than using jagtool or jagant to configure an entity's properties individually, you can define entity properties in XML. You can define and configure multiple entities in one XML file.

With jagtool, you can use the exportconfig command to create an XML configuration file that replicates an existing entity, and the configure command to apply the settings in an XML file to the repository.

You can also embed these XML configuration files in J2EE-format archive files for components, applications, Web applications, and connectors.

Note Support for this XML format is provided primarily for backward compatibility with EAServer 5.*x*. In 6.0 and later releases, use the Ant user-configuration target format described in Chapter 2, "Ant-Based Configuration."

Format of the XML configuration file

Configuration files must use this DTD, which is located in your EAServer installation:

```
lib/dtds/sybase-easerver-config_1_0.dtd
```

sybase-easerver-config element

The sybase-easerver-config element is the root element and can contain zero or more macro elements followed by one or more configure elements. You can use the description attribute for comment text; this text does not affect any properties set in the repository. Here is an example:

```
<sybase-easerver-config description="Configuring EAServer properties">
... deleted ...
</sybase-easerver-config>
```

macro elements

You can use macro elements to define abbreviations for commonly used strings in the XML file, such as the com.sybase.jaguar.component prefix used in most component property names. macro elements have optional begin and end attributes to specify the delimiters. If you do not specify them, the default begin and end delimiters are `${` and `}`, respectively. The macro element can contain one or more definition elements to specify the macro abbreviations, for example:

```
<macro begin="${" end="}">
    <definition name="comp" value="com.sybase.jaguar.component"/>
    <definition name="desc" value="com.sybase.jaguar.description"/>
</macro>
```

configure elements

A configure element creates and updates an entity. The value of the type attribute specifies the operation to perform, as follows:

Value of type attribute	Action
create	Create a new entity with the specified name and with the specified properties. The operation fails if the entity already exists.
update	Update the specified properties for an existing entity. The operation fails if the entity does not exist.
delete	Deletes the specified property settings from an existing entity. The operation fails if the entity does not exist.

If you are embedding an XML file in a J2EE archive to configure entities defined in the archive, use the update operation. Entities defined in the file exist when the XML configuration file is applied.

The entity attribute specifies the entity to operate on, using the format described in “Entity identifiers” on page 54. The configure element can contain zero or more property elements to configure the entity’s properties.

Here is an example configure element:

```
  <configure type="create" entity="Package:DocTest">
    <property name="${desc}" value="New package" />
  </configure>
```

property elements

The property element specifies a property setting for the entity, for example:

```
<property name="com.sybase.jaguar.component.stateless" value="true" />
```

Special characters

For special characters in property values, use the appropriate XML entity identifier, such as `&quot;` for the double-quote symbol (").

Sample configuration file

Here is a sample configuration file that defines a package, *DocTest*, and a component in the package, *FooComponent*:

```
<!DOCTYPE sybase-easerver-config PUBLIC '-
//Sybase, Inc.//DTD EAServer configuration 1.0//EN' 'http://www.sybase.com/dt
ds/easerver/sybase-easerver-config_1_0.dtd'>

<sybase-easerver-config description="Configuring EAServer properties">
  <macro begin="${" end="}">
        <definition name="comp" value="com.sybase.jaguar.component"/>
        <definition name="desc" value="com.sybase.jaguar.description"/>
  </macro>
  <configure type="create" entity="Package:DocTest">
    <property name="${desc}" value="New package" />
  </configure>
  <configure type="create" entity="Component:DocTest/FooComponent">
    <property name="${comp}.debug" value="true" />
    <property name="${comp}.name" value="DocTest/FooComponent" />
    <property name="${desc}" value="New description" />
    <property name="com.sybase.jaguar.component.type" value="java" />
    <property name="com.sybase.jaguar.component.control" value="JaguarEJB::Se
rverBean" />
    <property name="com.sybase.jaguar.component.sharing" value="true" />
    <property name="com.sybase.jaguar.component.roles" value="" />
    <property name="com.sybase.jaguar.component.tx_outcome" value="always" />
    <property name="com.sybase.jaguar.component.pooling" value="false" />
    <property name="com.sybase.jaguar.component.java.class" value="com.sybase
.jaguar.sample.events.StockManagerImpl" />
    <property name="com.sybase.jaguar.component.thread.safe" value="true" />
    <property name="com.sybase.jaguar.component.stateless" value="false" />
    <property name="com.sybase.jaguar.component.java.classes" value="" />
    <property name="com.sybase.jaguar.component.interfaces" value="EventSampl
es::StockManager" />
    <property name="com.sybase.jaguar.component.ids" value="IDL:EventSamples/
StockManager:1.0" />
    <property name="com.sybase.jaguar.component.transient" value="false" />
    <property name="com.sybase.jaguar.component.auto.failover" value="false"
/>
    <property name="com.sybase.jaguar.component.tx_type" value="not_supported
" />
    <property name="com.sybase.jaguar.component.tx_control" value="true" />
    <property name="com.sybase.jaguar.component.refresh" value="true" />
    <property name="com.sybase.jaguar.component.model" value="com" />
    <property name="com.sybase.jaguar.component.tx_vote" value="false" />
  </configure>
 </sybase-easerver-config>
```

jagtool commands

This section contains information about jagtool commands, and lists the commands that jagtool accepts.

Each command has its own heading and each command section contains a description of the command, its syntax, a list of options, and an example of its use at the command line and in Ant build files.

compilejsp

Description

Compiles JSP files.

Syntax

Local-mode support: Yes.

Connected-mode support: No.

Command line:

compilejsp
local-args
[-webapp *WebAppName*]
[-uriroot *directory*]
[-outputdir *directory*]
[-verbose *true*|*false*]
[-debug *true*|*false*]
[-keep *true*|*false*]
[*file1 file2 ...*]

Ant build file:

<jag_compilejsp
[webapp="*WebAppName*"]
[uriroot="*directory*"]
[outputdir="*directory*"]
[verbose="*true*|*false*"]
[debug="*true*|*false*"]
[keep="*true*|*false*"]
[jspfile="*file*"] />

Option	Description	Required
local-args	Arguments to run in local mode. See "Using local mode" on page 53.	Yes
`webapp`	Compiles the JSP files in the specified Web application. Uses the DJC_HOME environment variable to locate the Web application directory. Compiles all JSP files in the Web application unless specific files are specified. You must specify one of the `-webapp` or `-uriroot` options, but not both. You must specify the `-webapp` option to compile JSPs that require tag library mappings defined in the Web application properties file.	No
`uriroot`	Compiles the specified JSP files in the given directory, which must be specified as a complete path. If no files are specified, compiles all JSP files in the directory and its subdirectories.	No

Option	Description	Required
`outputdir`	The full path of the output directory for generated Java source and compiled classes. If the `-webapp` option is specified, the default is this subdirectory of your EAServer installation directory: `work\`*`server`*`\Servlet\WebApp-`*`WAName`* Where *server* is the name of your server, and *WAName* is the name of your Web application. To run the JSPs in EAServer, class files must be in this directory. If you do not specify the `-webapp` option, the default is the current directory.	No
`verbose`	Execute in verbose mode. The default is `false`.	No
`debug`	Compile the generated Java files with debugging information. The default is `false`.	No
`keep`	Keep the generated Java source files rather than deleting them. The default is `true`.	No
file1 file2 ...	When using the command line, the list of files to compile, with paths specified relative to the Web application root directory, if using the webapp option, or the specified directory, if using the uriroot option. If no files are specified, all files in the specified Web application or directory are compiled.	No
`jspfile`	When using Ant, the file to compile with path relative to the relative to the Web application root directory, if using the webapp option, or the specified directory, if using the uriroot option. If no file is specified, all files in the specified Web application or directory are compiled.	No

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This command line example compiles the files *jsp/file1.jsp* and *jsp/file2.jsp* in the Web application named MyWebApp:

```
jagtool compilejsp -webapp MyWebApp jsp/file1.jsp jsp/file2.jsp
```

The output is sent to the EAServer directory:

```
work/server/Servlet/WebApp-MyWebApp
```

Example 2 This command line example recursively compiles all JSPs in the Web application named MyWebApp:

```
jagtool compilejsp -webapp MyWebApp
```

The output is sent to the EAServer directory:

```
$DJC_HOME/work/server/Servlet/WebApp-MyWebApp
```

Where *server* is the name of the server to which you are connected.

Example 3 This command line example compiles *file1.jsp* in the Web application named MyWebApp:

```
jagtool compilejsp -webapp MyWebApp -outputdir c:\temp file1.jsp
```

The output is sent to *c:\temp*.

Example 4 This command line example compiles *file1.jsp* in the directory *c:\webapps\MyWebAppDir*:

```
jagtool compilejsp -uriroot c:\webapps\MyWebAppDir file1.jsp
```

The output is sent to the current directory.

Example 5 This command line example recursively compiles all JSPs in the directory *c:\webapps\MyWebAppDir*:

```
jagtool compilejsp -uriroot c:\webapps\MyWebAppDir -outputdir c:\temp
```

The output is sent to *c:\temp*.

Example 6 This Ant build file example defines a target to compile two JSP files:

```
<target name="compilejsp_test" >
  <jag_compilejsp Jspfile="file1.jsp" verbose="false"
Uriroot="D:\EAS\Sample\jagtool" />
  <jag_compilejsp Jspfile="file2.jsp" verbose="false"
Uriroot="D:\EAS\Sample\jagtool" />
</target>
```

Usage

You must run compilejsp in local mode. If you are running jagtool, see "Using local mode" on page 53. If you are running jagant, see "Using jagant in local mode" on page 61.

To use the compilejsp command in Ant, you must add the location of the *xalan.jar* and *crimson.jar* files that are included with EAServer to the Ant CLASSPATH. See "Setting up your environment" on page 56 for more information.

You can also compile JSPs using the EAServer jspc script. The jspc script invokes jagtool to compile JSPs.

See also

Chapter 4, "Creating JavaServer Pages," in the *EAServer Web Application Programmer's Guide*

configure

Description

Configures or defines entities in the repository by reading properties from an XML file.

Syntax

Local-mode support: Yes.

Command line:

```
configure
[ connect-args | local-args ]
[-verbose true|false]
filename
```

Ant build file:

```
<jag_configure [ verbose=”true|false” ] file=”filename” />
```

Where:

Option	Description	Required
connect-args | *local-args*	Arguments to specify a connection to the server or to run in local mode. See “Local versus connected mode” on page 52.	Yes
`verbose`	Execute in verbose mode. The default is `false`.	No
filename	XML file to read commands from.	Yes

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
1	The command ran successfully; the result is false/failure.
2	The command did not run successfully; an exception was thrown.

Usage

The configure command allows you to define and configure entities with an XML file.

Note In 6.0 and later releases, use the Ant user-configuration target format described in Chapter 2, “Ant-Based Configuration.”.

See also

exportconfig, “XML configuration files” on page 63

create

Description

Creates a new entity in the repository.

Syntax

Local-mode support: Yes.

Command line:

create [*connect-args* | *local-args*] *entity* [*file*]

Ant build file, specifying properties from an optional file:

<jag_create entity=”*entity*” [file=”*file*”] />

Ant build file, specifying properties directly:

```
<jag_create entity="entity" >
  <property name="name" value="value" />
   ....
</jag_create>
```

Option	Description	Required
connect-args | *local-args*	Arguments to specify a connection to the server or run in local mode. See “Local versus connected mode” on page 52.	Yes
entity	The name of the entity being created, in the form *EntityType:EntityName*.	Yes
file	An optional file containing properties for the entity. The file must specify properties in the form of an EAServer repository properties file.	No
name	The property name. In an Ant build file, you may specify multiple properties as `<property>` elements.	When setting properties directly in Ant
value	The property value.	When setting properties directly in Ant

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This example creates a package named NewPackage with the properties defined in the file *NewPackage.props*.

- Command line:

```
jagtool create Package:NewPackage NewPackage.props
```

- Ant build file:

```
<jag_create entity="Package:NewPackage" file="NewPackage.props" />
```

Example 2 This example creates a package named NewPackage2 and sets the given properties. This alternate syntax allows you to specify properties for the new entity in the command.

- Ant build file:

```
<jag_create entity="NewPackage2">
 <property name="com.sybase.jaguar.description" value=Sample Package" />
 <property name="com.sybase.jaguar.package.roles" value="role1" />
</jag_create>
```

Example 3 This example creates a listener called MyListener in the server called Jaguar:

- Command line:

  ```
  jagtool create Listener:Jaguar/MyListener
  ```

- Ant build file:

```
<jag_create entity="Listener:Jaguar/MyListener" />
```

Usage

The create command does not perform the installation steps required to run an entity in a particular server. For example, a package or application must be installed in a server before running on that server, and a listener must be installed in the server that it is associated with. Use the install command to install entities into a parent entity.

See also

delete, install, jmscreate

delete

Description

Deletes an entity from the repository.

Syntax

Local-mode support: Yes.

Command line:

delete [*connect-args* | *local-args*] [-type type] *entity*

Ant build file:

<jag_delete [type=”*type*”] entity=”*entity*” />

Option	Description	Required
connect-args | *local-args*	Arguments to specify a connection to the server or to run in local mode. See “Local versus connected mode” on page 52.	Yes
type	Specify `simple` for simple deletion (properties files only) or `full` to delete all files that were created when the entity was deployed. The default is `simple`. For more information, see “Undeploying entities” in Chapter 9, “Importing Application Components,” in the System Administration Guide.	No
entity	The identifier for the entity being deleted.	Yes

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
1	The command ran successfully; the result is false/failure.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This example deletes Package:SVU from the repository.

- Command line:

```
jagtool delete Package:SVU
```

- Ant build file:

```
<jag_delete entity="Package:SVU" />
```

Note When a package is deleted, it is also removed from any servers on which it is installed.

Example 2 This example deletes the SVULogin component from Package:SVU.

- Command line:

  ```
  jagtool delete Component:SVU/SVULogin
  ```

- Ant build file:

  ```
  <jag_delete entity="Component:SVU/SVULogin" />
  ```

deploy

Description

Deploys a J2EE EAR file, J2EE WAR file, J2EE JAR file, or EAServer JAR file to the server.

Syntax

Local-mode support: Yes.

Command line:

deploy
[*connect-args* | *local-args*]
[-type *filetype*]
[-stubsandskels *true*|*false*]
[-jagjartype *jartype*]
[-install *true*|*false*]
[-strategy *strategy*]
[-verbose *true*|*false*]
[-interoperablenaming *true*|*false*]
[-setjarfilepackagenaming=”*true*|*false*”]
filename

Ant build file:

<jag_deploy
[type="*filetype*"]
[stubsandskels="*true*|*false*"]
[jagjartype="*jartype*"]
[install="*true*|*false*"]
[strategy="*strategy*"]
[verbose="*true*|*false*"]
[interoperablenaming=”*true*|*false*”]
[setjarfilepackagenaming=”*true*|*false*”]
file="*filename*” />

Option	Description	Default	Required
connect-args | *local-args*	Arguments to specify a connection to the server or to run in local mode. See “Local versus connected mode” on page 52.	None	Yes
`type`	The type of file deployed: • ear • war • ejbjar • jagjar • rar	ear	No
`stubsandskels`	Indicates whether to generate stubs and skeletons for EJB files. This flag is applicable only to EAR and EJB files.	true	No

Option	Description	Default	Required
`jagjartype`	When *type* is `jagjar`, specifies the contents of the Jaguar JAR file. Options are: • Application • Connector • EntityCollection • Package • WebApplication	Application for type `jagjar`; otherwise, none	Only for type `jagjar`
`install`	Indicates whether to automatically deploy installed entities in servers.	true	No
`strategy`	Indicates the deployment strategy. Options are: • **full** IDL is generated for everything in the deployment tree. • **incremental** IDL is generated for everything that has changed. • **optimistic** Used for implementation changes (for example, if you have changed the contents of a method without changing its prototype). You can specify *incremental* when the methods, fields, interfaces, or superclass of a class have changed. Specify *full* when other details have changed or IDL has been deleted.	incremental	No
`setjarfilepackagenaming`	For EAR or EJB-JAR files, specifies how newly created packages are named, as follows: • `true` indicates that entities use the name of the archive file from which they are imported. • `false` indicates that entities use the `display-name` element in the deployment descriptor.	false	No
`verbose`	Indicates that output during deployment is verbose.	false	No
filename	The name of the deployed file.	None	Yes

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
1	The command ran successfully; the result is false/failure.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This command runs in local mode and deploys the EJB-JAR file named *myejb.jar* into the server named FooServer:

```
jagtool -local -server FooServer deploy -type ejbjar myejb.jar
```

Example 2 This example deploys the J2EE EAR file *eastore.ear* to the server:

- Command line:

```
jagtool deploy -type ear e:\temp\estore.ear
```

- Ant build file:

```
<jag_deploy type="ear" file="e:\temp\estore.ear" />
```

Example 3 This example deploys the JAR file *AuthServiceDemo.jar* to the server:

- Command line:

```
jagtool deploy -type jagjar -jagjartype Package /tmp/AuthServiceDemo.jar
```

- Ant build file:

```
<jag_deploy type="jagjar" jagjartype="Package"
file="/tmp/AuthServiceDemo.jar" />
```

See also

export, install

Chapter 9, “Importing Application Components” in the *System Administration Guide*.

ejbref

Description

Sets the value of an EJB reference.

Syntax

Local-mode support: Yes.

Command line:

ejbref
[*connect-args* | *local-args*]
entity
[-localref *true*|*false*]
-refname *name*
-value *value*

Ant build file:

<jag_ejbref [localref="*true*|*false*"] entity="*entity*" refname="*name*" value="*value*" />

Option	Description	Default	Required
connect-args | *local-args*	Arguments to specify a connection to the server or to run in local mode. See "Local versus connected mode" on page 52.	-	Yes
entity	The entity identifier in the form *EntityType*:*EntityName*.	-	Yes
`localref`	Whether setting a local-interface reference or a remote-interface reference. `true` indicates a local-interface reference.	`false`	No
`refname`	The name of an EJB reference for the given entity.	-	Yes
`value`	The value for the EJB reference.	-	Yes

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

This example sets the value of an EJB reference in the component TheCustomer in the package Customer_Component. The EJB reference ejb/account/Account is set to the value of "Customer_Component/TheAccount".

- Command line (all on one line):

```
jagtool ejbref Component:Customer_Component/TheCustomer
-refname ejb/account/Account
-value Customer_Component/TheAccount
```

- Ant build file:

```
<jag_ejbref entity="Component:Customer_Component/TheCustomer"
refname="ejb/account/Account" value="Customer_Component/TheAccount" />
```

enventry

Description

Sets the value of a J2EE environment entry.

Syntax

Local-mode support: Yes.

Command line:

```
enventry
[ connect-args | local-args ]
entity
-entryname name
-value value
```

Ant build file:

```
<jag_enventry entity=”entity” entryname=”name” value=”value” />
```

Option	Description	Default	Required
connect-args | *local-args*	Arguments to specify a connection to the server or to run in local mode. See “Local versus connected mode” on page 52.	-	Yes
entity	The entity identifier in the form *EntityType*:*EntityName*.	-	Yes
`entryname`	The name of an environment entry for the given entity.	-	Yes
`value`	The value for the environment entry.	-	Yes

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

This example sets the value of an environment entry in the component TheOrder in the package Customer_Component. The value of environment entry ejb/order/OrderDAOClass is set to “com.sun.j2ee.blueprints.customer.order.dao.OrderDAOSybase”.

- Command line (all on one line):

```
jagtool enventry Component:Customer_Component/TheOrder -entryname
ejb/order/OrderDAOClass -value
com.sun.j2ee.blueprints.customer.order.dao.OrderDAOSybase
```

- Ant build file:

```
<jag_enventry entity="Component:Customer_Component/TheOrder"
entryname="ejb/order/OrderDAOClass"
value="com.sun.j2ee.blueprints.customer.order.dao.OrderDAOSybase" />
```

exists

Description

Determines whether or not a specified entity is present in the configuration repository.

Syntax

Local-mode support: Yes.

Command line:

exists [*connect-args* | *local-args*] *entity*

Ant build file:

<jag_exists entity=”*entity*” property=”*ant_property*”>

Option	Description	Required
connect-args | *local-args*	Arguments to specify a connection to the server or to run in local mode. See “Local versus connected mode” on page 52.	Yes
entity	The entity identifier in the form *EntityType*:*EntityName*.	Yes
ant_property	The name of the Ant build property to set if the entity exists.	When using Ant

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
1	The command ran successfully; the result is false/failure.
2	The command did not run successfully; an exception was thrown.

Examples

This example determines whether the component SVULogin in package SVU exists:

- Command line:

```
jagtool exists Component:SVU/SVULogin
```

- Ant build file:

  This example does the same, and sets the property `svulogin.exists` if the component exists. If it does not exist, the property is not set.

```
<jag_exists entity="Component:SVU/SVULogin" property="svulogin.exists"
/>
```

export

Description

Exports an entity to a J2EE-format EAR, JAR, RAR, or WAR file, or to an EAServer-format JAR file.

Syntax

Local-mode support: Yes.

Command line:

```
export [ connect-args | local-args ] [-dir dirname] [-jagjar true|false] \
  [-xmlconfig=true|false] [-emptycachetags=true|false] entity
```

Ant build file:

```
<jag_export [ dir=”dirname”] [ jagjar=”true|false”]
[-xmlconfig=”true|false”] -[emptycachetags=”true|false”]
entity=”entity” />
```

Option	Description	Default	Required
connect-args | *local-args*	Arguments to specify a connection to the server or to run in local mode. See “Local versus connected mode” on page 52.	-	Yes
`dir`	The directory where the file is created.	Current directory	No
`jagjar`	Export to a Jaguar JAR file rather than a J2EE archive file.	true	No
`xmlconfig`	When exporting a J2EE archive, whether to include the EAServer-specific *sybase-easerver-config.xml* file. For more information, see “Using EAServer configuration files in J2EE archives” in Chapter 9, “Importing Application Components” in the *System Administration Guide*.	true	No
`emptycachetags`	When exporting a J2EE archive, whether to export a no-operation version of the EAServer partial page caching tag library with Web applications to allow portability to other J2EE application servers that do not support this tag library.	false	No
entity	Entity identifier for the entity being exported.	-	Yes

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
1	The command ran successfully; the result is false/failure.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This example exports the application named “estore” to a J2EE EAR file in the *e:\temp* directory:

- Command line:

```
jagtool export -dir e:\temp Application:estore
```

- Ant build file:

```
<jag_export dirname="e:\temp" entity="Application:estore" />
```

Example 2 This example exports the SVU package to a JAR file in the current directory:

- Command line:

```
jagtool export -jagjar true Package:SVU
```

- Ant build file:

```
<jag_export jagjar="true" entity="Package:SVU" />
```

See also

deploy

Chapter 9, “Importing Application Components” in the *System Administration Guide*.

exportconfig

Description

Creates an XML configuration file that matches the configuration of an existing entity.

Syntax

Local-mode support: Yes.

Command line:

```
exportconfig [ connect-args | local-args ] [-dir dirname] \
  [-configtype update|create] [-verbose true|false] \
  [-easerverpropsonly true|false ] entity
```

Ant build file:

```
<jag_exportconfig [ dir=”dirname”] [configtype=”update|create”]
[verbose=true|false”] [easerverpropsonly “true|false “]
entity=”entity” />
```

Option	Description	Default	Required
connect-args | *local-args*	Arguments to specify a connection to the server or to run in local mode. See “Local versus connected mode” on page 52.	-	Yes
`dir`	The directory where the file is to be created.	Current directory	No
`configtype`	Specifies the value of the type attribute of the configure XML element that is created.	`update`	No
`verbose`	Execute in verbose mode.	`false`	No
`easerverpropsonly`	Whether the generated XML file should include all properties, or only those properties that are not set in an equivalent J2EE deployment descriptor for the entity. Specify true to exclude properties that are set in the equivalent J2EE deployment descriptor.	`false`	No
entity	The name of the entity being created, in the form *EntityType:EntityName*.	-	Yes

The exportconfig command creates the file *sybase-easerver-config.xml* in the specified directory. The command fails if a file with this name already exists.

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
1	The command ran successfully; the result is false/failure.
2	The command did not run successfully; an exception was thrown.

Examples

This command line example creates an XML configuration file in the current directory for the component *EventSamples/StockManager* and specifies `create` as the configure type:

```
jagtool exportconfig -configtype create Component:EventSamples/StockManager
```

Usage

In 6.0 and later releases, use the Ant user-configuration target format described in Chapter 2, “Ant-Based Configuration.”

See also

configure, “XML configuration files” on page 63

getmonitorstats

Description

Retrieves and prints runtime monitoring statistics from the server to which you are connected.

Syntax

Local-mode support: No.

Command line:

getmonitorstats *connect-args item* [*detail*]

Ant build file:

<jag_getmonitorstats [item="*item*"] [detail="*detail*"] />

Option	Description	Required
connect-args	Arguments to specify a connection to the server. See "Using connected mode" on page 52.	Yes
item	An item type as listed in Table 6-4.	Yes
detail	An optional detail specifier to match the specified item type, as listed in Table 6-4.	No

Usage

getmonitorstats allows you to retrieve runtime monitoring statistics for the items listed in Table 6-4.

Table 6-4: getmonitorstats options

Item	Detail	To specify
Package	An optional package name	Statistics for components in the specified package, or all components if you do not specify a package name
Component	A component name in the form `package/component`	Statistics for the specified component
ConnCache	An optional connection cache name	Statistics for the specified connection cache, or all of them if you do not specify a name
ManagedConnectionFactory	An optional managed connection factory name	Statistics for the specified managed connection factory, or all of them if you do not specify a name
Network	The protocol, that is, `HTTP` or `IIOP`	Network statistics for the specified protocol

See also

Chapter 11, "Runtime Monitoring" in the *System Administration Guide*.

getserverinfo

Description

Print status and version information for the server to which you are connected.

Syntax

Local-mode support: No.

Command line:

```
getserverinfo connect-args [-version true|false] [-status true|false]
```

Ant build file:

```
<jag_getserverinfo [ version="true|false" ] [ status="true|false" ] />
```

Option	Description	Required
connect-args	Arguments to specify a connection to the server. See "Using connected mode" on page 52.	Yes
`version`	Whether to print the server version number.	No
`status`	Whether to print the server status, that is, whether the server is accepting regular client connections or in Admin mode.	No

getservicestate

Description

Returns the state of service components executing in the server.

Syntax

Local-mode support: No.

Command line:

```
getservicestate connect-args servicename
```

Ant build file:

```
<jag_getservicestate servicename="servicename " />
```

Option	Description	Required
connect-args	Arguments to specify a connection to the server. See "Using connected mode" on page 52.	Yes
servicename	The name of the service to query, or all to list the state of all services. jagtool fails with an error message if you specify the name of a service that is not installed or that does not implement the interface `CtsServices::ExtendedService`.	Yes

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
1	The command ran successfully; the result is false/failure.
2	The command did not run successfully; an exception was thrown.

Usage

This command prints the service state, using the terms listed in Table 6-5.

Table 6-5: Service states

State	Description
`UNKNOWN`	The state is unknown because the service does not implement the `CtsServices::ExtendedService` interface.
`STARTING`	The service is starting. The start method has been called, but has not returned.
`STARTED`	The service is started, but not yet running. The start method has returned, but the run method has not been called.
`RUNNING`	The service is running, that is, executing the run method.

State	Description
`FINISHED`	The service is finished processing. The run method has returned. This state applies only to services that do not run continuously until stopped.
`STOPPING`	The service is stopping. The stop method has been called, and is still running.
`STOPPED`	The service is stopped. The stop method has been called and has returned.

For example, this is the typical output for a server where the message service is not installed:

```
jagtool getservicestate all
JaguarServlet/ServletService's state is RUNNING
CosNaming/JNameService's state is FINISHED
```

This command is useful when you are running jagtool or jagant from scripts that start or restart the server. You can check the output to determine if the required service is running. For example, components should not be deployed before the name service has finished, and JMS entities cannot be created before the message service is running.

See also

Chapter 4, "Creating Service Components"

grantroleauth

Description

Grants authorization to a given role to perform specific actions on the given entity. If the entity is a server, members of the role are granted permission to restart, refresh, or shutdown the server. If the entity is an application, Web application, servlet, or package, members of the role are granted access to those resources, including deploying the entity.

Syntax

Local-mode support: Yes.

Command line:

grantroleauth [*connect-args* | *local-args*] [-role *rolename*] \
  [-action *actionname*] *entity*

Ant build file:

<jag_grantroleauth [role=”*rolename*”] [action=”*actionname*”]
entity=”*entity*” />

Option	Description	Required
connect-args | *local-args*	Arguments to specify a connection to the server or to run in local mode. See “Local versus connected mode” on page 52.	Yes
`role`	The role ID or name to which authorization is being granted. The role must exist on the server to which you are connected.	Yes
`action`	Valid only when the entity type is server. Valid actions include restart, refresh, and shutdown.	When the entity type is server
entity	The name of the entity, in the form *EntityType:EntityName*. Valid entities are application, Webapplication, servlet, server, and package.	Yes

Examples

This example grants access to the “Estore” application to members of the role named “test”.

```
jagtool grantroleauth -role test Application:Estore
```

See also

removeroleauth

install

Description

Installs an entity into another entity (for example, installs a package into a server).

Syntax

Local-mode support: Yes.

Command line:

install [*connect-args* | *local-args*] *source target*

Ant build file:

<jag_install source=”*source*” target=”*target*” />

Option	Description	Required
connect-args | *local-args*	Arguments to specify a connection to the server or to run in local mode. See “Local versus connected mode” on page 52.	Yes
source	The entity identifier for the entity being installed.	Yes
target	The entity identifier in which the source entity is being installed.	Yes

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
1	The command ran successfully; the result is false/failure.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This example installs the package SoapDemo in the server Jaguar.

- Command line:

```
jagtool install Package:SoapDemo Server:Jaguar
```

- Ant build file:

```
<jag_install source="Package:SoapDemo" target="Server:Jaguar" />
```

Example 2 This example installs the application MyPortfolio in the server Jaguar.

- Command line:

```
jagtool install Application:MyPortfolio Server:Jaguar
```

- Ant build file:

```
<jag_install source="Application:MyPortfolio" target="Server:Jaguar" />
```

Example 3 This example installs MyListener into the server Jaguar:

- Command line:

```
jagtool install Listener:Jaguar/MyListener Server:Jaguar
```

- Ant build file:

```
<jag_install source="Listener:Jaguar/MyListener" target=”Server:Jaguar”
/>
```

Example 4 This example installs the service component MyPack/MyComp into the server Jaguar:

- Command line:

```
jagtool install Service:MyPack/MyComp Server:Jaguar
```

- Ant build file:

```
<jag_install source="Service:MyPack/MyComp" target=”Server:Jaguar” />
```

See also create, remove

jmscreate

Description

Creates a JMS entity.

Syntax

Local-mode support: No.

Command line:

jmscreate *connect-args entity* [*file*]

Ant build file:

<jag_jmscreate entity=”*entity*” [file=”*file*”] />

Option	Description	Required
connect-args	Arguments to specify a connection to the server. See “Using connected mode” on page 52.	Yes
entity	The entity to create, in the form `EntityType:EntityName`. Valid entity types are: • MessageQueue • MessageTopic • QueueConnectionFactory • ThreadPool • TopicConnectionFactory	Yes
file	An optional file containing properties for the entity. The file must specify properties in the form of an EAServer repository properties file. You can set properties after the entity exists with the jmsset_props command.	No

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This example uses the command line to create a message queue named MyQueue and specifies a properties file:

```
jagtool jmscreate MessageQueue:MyQueue
D:\Jag41005\sample\jagtool\queueprops.txt
```

Example 2 This example does the same thing in an Ant build file:

```
<jag_jmscreate entity="MessageQueue:AntQueue"
file="D:\Jag41005\sample\jagtool\queueprops.txt" />
```

See also

jmsdelete, jmslist, jmsprops, jmsset_props

Chapter 2, “Setting up the Message Service” the EAServer JMS User’s Guide.

jmsdelete

Description

Deletes the specified entity.

Syntax

Local-mode support: No.

Command line:

jmsdelete *connect-args entity*

Ant build file:

<jag_jmsdelete entity=”*entity*” />

Option	Description	Required
connect-args	Arguments to specify a connection to the server. See “Using connected mode” on page 52.	Yes
entity	The entity to delete, in the form `EntityType:EntityName`. Valid entity types are: • Listener • MessageQueue • MessageTopic • QueueConnectionFactory • ThreadPool • TopicConnectionFactory	Yes

If the entity is a message queue, you must first remove its listeners. This operation removes selectors, and receives and acknowledges all queued messages for the message queue.

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This command line example deletes the message queue “MyQueue”:

```
jagtool jmsdelete MessageQueue:MyQueue
```

Example 2 This example does the same thing in an Ant build file:

```
<jag_jmsdelete entity="MessageQueue:MyQueue" />
```

jmsflush

Description

Flushes the messages from the specified message queue.

Syntax

Local-mode support: No.

Command line:

jmsflush *connect-args* MessageQueue:*QueueName*

Where *connect-args* is the list of arguments to specify a connection to the server. See “Using connected mode” on page 52.

Ant build file:

<jag_jmsflush entity=”MessageQueue:*QueueName*” />

Return value

Return value	**Indicates**
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This command line example flushes the message queue “MyQueue”:

```
jagtool jmsflush MessageQueue:MyQueue
```

Example 2 This example does the same thing in an Ant build file:

```
<jag_jmsflush entity="MessageQueue:MyQueue" />
```

jmslist

Description

Lists JMS entities of the specified type.

Syntax

Local-mode support: No.

Command line:

jmslist *connect-args type*

Ant build file:

<jag_jmslist type="*type*" />

Where *connect-args* is the list of arguments to specify a connection to the server, described in "Using connected mode" on page 52, and *type* is one of the following JMS entity types:

Type specifier	To list names of
`ActiveMessageQueue`	Currently active message queues
`ConfMessageQueue`	Configured message queues (queues that were created administratively rather than programmatically)
`ActiveMessageTopic`	Currently active message topics
`ConfMessageTopic`	Configured message topics (topics that were created administratively rather than programmatically)
`QueueConnectionFactory`	Queue connection factories
`ThreadPool`	Thread pools
`TopicConnectionFactory`	Topic connection factories

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This command line example lists active JMS message queues:

```
jagtool jmslist ActiveMessageQueue
```

Example 2 This example does the same in an Ant build file:

```
<jag_jmslist type="ActiveMessageQueue" />
```

jmslist_listeners

Description

Lists the names of the listeners attached to the specified message queue.

Syntax

Local-mode support: No.

Command line:

jmslist_listeners *connect-args* MessageQueue:*QueueName*

Where *connect-args* is the list of arguments to specify a connection to the server. See "Using connected mode" on page 52.

Ant build file:

<jag_jmslist_listeners type="MessageQueue:*QueueName*" />

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This command line example lists the listeners attached to "MyQueue":

```
jagtool jmslist_listeners MessageQueue:MyQueue
```

Example 2 This example does the same in an Ant build file:

```
<jag_jmslist_listeners type="MessageQueue:MyQueue"
/>
```

jmslist_messages

Description

Lists the messages in the specified message queue.

Syntax

Local-mode support: No.

Command line:

jmslist_messages *connect-args* [-maximum *#messages*]
[-selector *expression*]
MessageQueue:*QueueName*

Ant build file:

<jag_jmslist_messages [“maximum = *#messages*]
[“selector = *expression*”]
type=”MessageQueue:*QueueName*” />

Option	Description	Default	Required
connect-args	Arguments to specify a connection to the server. See “Using connected mode” on page 52.	-	Yes
#messages	The maximum number of messages to list. If 0 or a negative number, all messages are listed.	100	No
expression	A selector expression. Only messages that match the selector expression are returned. If set to true, all messages are returned. The number of these messages that are listed is determined by *#messages*.	true	No

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This command line example lists the first 25 messages in “MyQueue”:

```
jagtool jmslist_messages -maximum 25 MessageQueue:MyQueue
```

Example 2 This example does the same in an Ant build file:

```
<jag_jmslist_messages “maximum=25” type="MessageQueue:MyQueue" />
```

jmsmanage_listeners

Description

Adds and removes listeners to and from JMS message queues.

Syntax

Local-mode support: No.

Command line:

jmsmanage_listeners
connect-args
-*action* "Component:*comp*"
MessageQueue:*queue*

Ant build file: <

jag_manage_listeners action=”*action*” listener="Component:*comp*"
entity="MessageQueue:*queue*"/>

Option	Description	Required
connect-args	Arguments to specify a connection to the server. See “Using connected mode” on page 52.	Yes
action	`add` to add the listener to the queue or `remove` to remove the listener from the queue.	Yes
comp	The component that listens for messages on the specified queue, in the format *`package/component`*.	Yes
queue	The message queue name.	Yes

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

These command line examples add and remove the component named JmsListenerTest/JmsListener to the queue MyQueue:

```
jagtool jmsmanage_listeners -add Component:JmsListenerTest/JmsListener
MessageQueue:MyQueue

jagtool jmsmanage_listeners -remove Component:JmsListenerTest/JmsListener
MessageQueue:MyQueue
```

jmsmanage_selectors

Description

Adds and removes selectors to and from JMS message queues.

Syntax

Local-mode support: No.

Command line:

jmsmanage_selectors
connect-args
-action "*selector*"
MessageQueue:*queue*

Ant build file:

<jag_manage_selectors action=”*action*” selector="*selector*"
entity="MessageQueue:*queue*"/>

Option	Description	Required
connect-args	Arguments to specify a connection to the server. See “Using connected mode” on page 52.	Yes
action	`add` to add the selector to the queue or `remove` to remove the selector from the queue.	Yes
selector	The JMS selector.	Yes
queue	The message queue name.	Yes

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

These command line examples install and remove the selector “topic='test'” from the message queue MyQueue:

```
jagtool jmsmanage_selectors -add "topic='test'" MessageQueue:MyQueue

jagtool jmsmanage_listeners -remove "topic='test'" MessageQueue:MyQueue
```

jmsprops

Description

Lists the properties for a JMS entity.

Syntax

Local-mode support: No.

Command line:

jmsprops *connect-args entity*

Ant build file:

<jag_jmsprops entity=”*entity*” />

Option	Description	Required
connect-args	Arguments to specify a connection to the server. See “Using connected mode” on page 52.	Yes
entity	The entity of interest, in the form `EntityType:EntityName`. Valid entity types are: • MessageQueue • MessageTopic • QueueConnectionFactory • ThreadPool • TopicConnectionFactory	Yes

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This example lists properties for the message queue “MyQueue”:

```
jagtool jmsprops MessageQueue:MyQueue
```

Example 2 This example does the same thing in an Ant build file:

```
<jag_jmsprops entity="MessageQueue:MyQueue" />
```

See also

jmsset_props

jmsset_props

Description

Sets properties for a JMS entity.

Syntax

Local-mode support: No.

Command line: To set an individual property, specify the property name and value on the command line:

```
jmsset_props connect-args entity name value
```

To set multiple properties (one or more) from a properties file, specify the file name:

```
jmsset_props connect-args entity file
```

Ant build file: To set an individual property, specify the property name and value:

```
<jag_jmsset_props entity=”entity” name=”name” value=”value” />
```

To set multiple properties (one or more) from a properties file, specify the file name:

```
<jag_jmsset_props entity=”entity” file=”file” />
```

Option	Description	Required
connect-args	Arguments to specify a connection to the server. See “Using connected mode” on page 52.	Yes
entity	The entity of interest, in the form `EntityType:EntityName`. Valid entity types are: • MessageQueue • MessageTopic • QueueConnectionFactory • ThreadPool • TopicConnectionFactory	Yes
name	The property name.	No
value	The property value.	No
file	An optional file containing properties for the entity. The file must specify properties in the form of an EAServer repository properties file.	No

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This command line example configures the "maximum" property of the queue named "AntQueue":

```
jagtool jmsset_props "MessageQueue:AntQueue" maximum 99
```

This example does the same thing in an Ant build file:

```
<jag_jmsset_props entity="MessageQueue:AntQueue" name="maximum" value="99"
/>
```

Example 2 This command line example configures the queue named "AntQueue," specifying a properties file:

```
jagtool jmsset_props "MessageQueue:AntQueue"
"D:\Jag41005\sample\jagtool\Newqueueprops.txt"
```

This example does the same thing in an Ant build file:

```
<jag_jmsset_props entity="MessageQueue:AntQueue"
file="D:\Jag41005\sample\jagtool\Newqueueprops.txt" />
```

Example 3 Here is what the *Newqueueprops.txt* file used in the above examples might contain:

```
IGNORE_DUPLICATE_KEY=false
REQUIRES_ACKNOWLEDGE=false
REQUIRES_TRANSACTION=false
maximum=0
qop=none
share=true
store=true
table=
timeout=60
```

See also

jmscreate, jmsdelete, jmslist, jmsprops

Chapter 2, "Setting up the Message Service" in the *JMS User's Guide*

list

Description

Lists entities in the repository.

Syntax

Local-mode support: Yes.

Command line:

list [*connect-args* | *local-args*] *type entity*

Ant build file:

<jag_list type="*type*" entity="*entity*" />

Option	Description	Required
connect-args | *local-args*	Arguments to specify a connection to the server or to run in local mode. See "Local versus connected mode" on page 52.	Yes
type	The type of entities to list.	Either *type* or *entity* is required. Both can be used.
entity	An optional entity identifier to specify a parent entity. Child entities are listed.	

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This command lists all connection caches, running from the command line in local mode:

```
jagtool -local list ConnCache
```

Example 2 This example lists all the packages in the repository.

- Command line:

```
jagtool list Package
```

- Ant build file:

```
<jag_list type="Package" />
```

Example 3 This example lists all the child entities of Package:SVU.

- Command line:

```
jagtool list Package:SVU
```

- Ant build file:

```
<jag_list entity="Package:SVU" />
```

Example 4 This example lists all the child components of Package:SVU.

- Command line:

  ```
  jagtool list Component Package:SVU
  ```

- Ant build file:

```
<jag_list type="Component" entity="Package:SVU" />
```

merge_props

Description

Merges or deletes property values for an entity.

Syntax

Local-mode support: Yes.

Command line:

```
merge_props [ connect-args | local-args ] entity \
   [-verbose true|false] { [mergeop name value] | [file] }
```

Ant build file: There are three syntax forms for Ant commands. You can specify a merge command for a single property with this syntax:

```
<jag_merge_props entity=”entity” [ verbose=” true|false” ]
mergeop=”mergeop” name=”name” value=”value” >
</jag_merge_props>
```

You can specify merge commands for multiple properties with this syntax:

```
<jag_merge_props entity="entity" [ verbose="true|false" ]>
  <mergeproperty mergeop="mergeop" name="name"
     value="value" />
  <mergeproperty mergeop="mergeop" name="name"
     value="value" />
  ...
</jag_merge_props>
```

You can specify the name of a file that contains merge commands with this syntax:

```
<jag_merge_props entity=”entity” [ verbose=” true|false” ]
  file=”file” ></jag_merge_props>
```

Option	Description	Required
connect-args | *local-args*	Arguments to specify a connection to the server or to run in local mode. See “Local versus connected mode” on page 52.	Yes
entity	The entity identifier for the target entity. Valid entity types include Agent, Application, Cluster, Component, ConnCache, Connector, DatabaseType, EntityCollection, Filter, InstancePool, Listener, ManagedConnectionFactory, Package, Role, Security, Server, Servlet, and WebApplication. The entity types Connector, Filter, and ManagedConnectionFactory are available only for J2EE 1.3-enabled servers.	Yes
`verbose`	Whether to execute in verbose mode. The default is `false`.	No

Option	Description	Required
mergeop	The merge operation, one of: • **AppendToList** For properties that take a comma-separated list of values, append the value to the existing value if not already present. Case is significant when comparing against existing entries; for example, if foo2 is present and you append Foo2, both are present after appending. • **PrependToList** For properties that take a comma-separated list of values, prepend the value to the existing value. • **RemoveFromList** For properties that take a comma-separated list of values, remove the value from the existing value if present. • **SetDefault** Set the properties value to the default value. • **Delete** Delete the property setting completely.	When not specifying a file containing merge commands
name	The name of the property of interest.	When not specifying a file containing merge commands
value	The property value to merge.	When not specifying a file containing merge commands
file	The name of a text file containing merge commands. The file must be a text file containing lines of the form: *mergeop*:*name*=*value* Where *mergeop*, *name*, and *value* follow the syntax rules above.	When not specifying a merge operation and property name

Examples

Example 1 To add a service to the list of services for the server named Jaguar, you can run the following on the command line:

```
jagtool merge_props Server:Jaguar AppendToList
com.sybase.jaguar.server.services MyNewService
```

Example 2 The following Ant example appends a value to the Java classes setting for a package:

```
<target name="test_merge_props" depends="connect">
<jag_merge_props entity="Package:Foo"
  verbose="true" mergeop="AppendToList"
  name="com.sybase.jaguar.package.java.classes"
  value="com.foo.MyClass">
</jag_merge_props>
</target>
```

Example 3 This Ant example merges several properties for a package:

```
<target name="test_merge_props" depends="connect">
<jag_merge_props entity="Package:Foo" verbose="true">
  <mergeproperty mergeop="AppendToList"
    name="com.sybase.jaguar.package.description" value="more stuff"/>
  <mergeproperty mergeop="PrependToList"
    name="com.sybase.jaguar.package.someprop" value="newvalue3"/>
  <mergeproperty mergeop="RemoveFromList"
    name="com.sybase.jaguar.package.someprop" value="original"/>
  <mergeproperty mergeop="SetDefault"
    name="com.sybase.jaguar.package.someprop" value="newvalue4"/>
  <mergeproperty mergeop="Delete"
    name="com.sybase.jaguar.package.someprop"/>
</jag_merge_props>
</target>
```

Example 4 This Ant example specifies the name of a file that contains merge commands:

```
<target name="test_merge_props" depends="connect">
<jag_merge_props entity="Package:Foo"
  verbose="true" file="C:\EAServer\MergeProps.txt">
</jag_merge_props>
</target>
```

See also

props, set_props

ping

Description

Pings a connection cache.

Syntax

Local-mode support: No.

Command line:

ping *connect-args entity*

Ant build file:

<jag_ping entity=”*entity*” />

Option	Description	Required
connect-args	Arguments to specify a connection to the server. See “Using connected mode” on page 52.	Yes
entity	The connection cache identifier in the form `ConnCache:`*`EntityName`*.	Yes

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
1	The command ran successfully; the result is false/failure.
2	The command did not run successfully; an exception was thrown.

Examples

This example pings the connection cache named “JavaCache”:

- Command line:

```
jagtool ping ConnCache:JavaCache
```

- Ant build file:

```
<jag_ping entity="ConnCache:JavaCache" />
```

props

Description

Lists properties for an entity in the repository.

Syntax

Local-mode support: Yes.

Command line:

props [*connect-args* | *local-args*] *entity*

Ant build file:

<jag_props entity=”*entity*” />

Option	**Description**	**Required**
connect-args | *local-args*	Arguments to specify a connection to the server or to run in local mode. See “Local versus connected mode” on page 52.	Yes
entity	The entity identifier for the entity whose properties are listed.	Yes

Return value

Return value	**Indicates**
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This example lists the properties of Package:SVU.

- Command line:

```
jagtool props Package:SVU
```

- Ant build file:

```
<jag_props entity="Package:SVU" />
```

Example 2 This example lists the properties of Server:Jaguar.

- Command line:

```
jagtool props Server:Jaguar
```

- Ant build file:

```
<jag_props entity="Server:Jaguar" />
```

See also

Appendix B, “Repository Properties Reference”

rebind

Description

Rebinds a cluster, which refreshes all of the name servers within the cluster.

Syntax

Local-mode support: No.

Command line:

rebind *connect-args* Cluster:*name*

Ant build file:

<jag_rebind entity="Cluster:*name*" />

Option	Description	Required
connect-args	Arguments to specify a connection to the server. See "Using connected mode" on page 52.	Yes
name	The name of the cluster to rebind.	Yes

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Usage

If you add a component to a server that is already part of a cluster and want to make that component available to the cluster, you need to rebind the cluster. You can also use the rebind option if a problem occurs when you synchronize the cluster; if for example, one of the name servers is slow to start.

See also

Chapter 6, "Clusters and Synchronization" in the *System Administration Guide*.

refresh

Description

Refreshes an entity in the server.

Syntax

Local-mode support: No.

Command line:

refresh *connect-args entity*

Ant build file:

<jag_refresh entity="*entity*" />

Option	Description	Required
connect-args	Arguments to specify a connection to the server. See “Using connected mode” on page 52.	Yes
entity	The entity identifier for the entity being refreshed. To refresh servers, you must be connected to the specified server. To refresh other entities, you must be connected to a server in which the entity is installed. Refresh does not support servlets. To refresh servlets, refresh the Web application or server in which they are installed.	Yes

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This example refreshes Package:SVU.

- Command line:

```
jagtool refresh Package:SVU
```

- Ant build file:

```
<jag_refresh entity="Package:SVU" />
```

Example 2 This example refreshes the SVULogin component of Package:SVU.

- Command line:

```
jagtool refresh Component:SVU/SVULogin
```

- Ant build file:

```
<jag_refresh entity="Component:SVU/SVULogin" />
```

Example 3 This example refreshes the server *Jaguar*, and works only when you are connected to the server with this name.

- Command line:

```
jagtool refresh Server:Jaguar
```

- Ant build file:

```
<jag_refresh entity="Server:Jaguar" />
```

remove

Description

Removes, but does not delete, an entity from another entity. For example, use remove to remove a package from a server.

Syntax

Local-mode support: Yes.

Command line:

remove [*connect-args* | *local-args*] *source target*

Ant build file:

<jag_remove source="*source*" target="*target*" />

Option	Description	Required
connect-args | *local-args*	Arguments to specify a connection to the server or to run in local mode. See "Local versus connected mode" on page 52.	Yes
source	The entity identifier of the entity being removed.	Yes
target	The entity identifier of the entity from which the *source* is removed.	Yes

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
1	The command ran successfully; the result is false/failure.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This example removes Package:SVU from the entity Server:Jaguar.

- Command line:

```
jagtool remove Package:SVU Server:Jaguar
```

- Ant build file:

```
<jag_remove source="Package:SVU" target="Server:Jaguar"/>
```

Example 2 This example removes WebApplication:WebTier from the entity Application:estore.

- Command line:

```
jagtool remove WebApplication:WebTier Application:estore
```

- Ant build file:

```
<jag_remove source="WebApplication:WebTier" target="Application:estore"
/>
```

Example 3 This example removes the service component MyPack/MyComp from the server Jaguar:

- Command line:

```
jagtool remove Service:MyPack/MyComp Server:Jaguar
```

- Ant build file:

```
<jag_remove source="Service:MyPack/MyComp" target=”Server:Jaguar” />
```

See also install

removeroleauth

Description

Removes authorization from members of a given role to perform specific actions on the given entity. If the entity is a server, members of the role are denied permission to restart, refresh, or shut down the server. If the entity is an application, Web application, servlet, or package, members of the role are denied access to those resources, including deploying the entity.

Syntax

Local-mode support: Yes.

Command line:

removeroleauth [*connect-args* | *local-args*] [-role *rolename*] \
  [-action *actionname*] *entity*

Ant build file:

<jag_removeroleauth [role="*rolename*"] [action="*actionname*"] entity="*entity*" />

Option	Description	Required
connect-args | *local-args*	Arguments to specify a connection to the server or to run in local mode. See "Local versus connected mode" on page 52.	Yes
`role`	Role ID or name to which authorization is being denied. The role must exist on the server to which you are connected.	Yes
`action`	Only valid when the entity type is server. Valid actions include restart, refresh, and shutdown.	No
entity	The name of the entity, in the form *EntityType:EntityName*. Valid entities are application, Webapplication, servlet, server, and package.	Yes

Examples

This example denies access to the "Estore" application to members of the role named "test".

```
jagtool removeroleauth -role test Application:Estore
```

See also

grantroleauth

resref

Description

Sets the value of a J2EE resource reference.

Syntax

Local-mode support: Yes.

Command line:

resref [*connect-args* | *local-args*] *entity* -refname *name* -value *value*

Ant build file:

<jag_resref entity=”*entity*” refname=”*name*” value=”*value*” />

Option	Description	Required
connect-args | *local-args*	Arguments to specify a connection to the server or to run in local mode. See “Local versus connected mode” on page 52.	Yes
entity	The entity identifier in the form *EntityType*:*EntityName*.	Yes
name	The name of a resource reference for the given entity.	Yes
value	The value for the reference.	Yes

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

This example sets the value of a reference in the component TheAccount in the Customer_Component package. The resource reference jdbc/EstoreDataSource is set to the value “PetStoreDB.”

- Command line (all on one line):

```
jagtool resref Component:Customer_Component/TheAccount -refname
jdbc/EstoreDataSource -value PetStoreDB
```

- Ant build file:

```
<jag_resref entity="Component:Customer_Component/TheAccount"
refname="jdbc/EstoreDataSource" value="PetStoreDB" />
```

restart

Description

Terminates and restarts the server process to which you are connected.

Syntax

Local-mode support: No.

Command line:

restart *connect-args*

Where *connect-args* is a list of arguments to specify a server connection, as described in "Using connected mode" on page 52.

Ant build file:

<jag_restart />

Return value

Return value	**Indicates**
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

This example connects to the server "eclipse" on port 9005, with the user name "admin@system" and the password "jagpass," and restarts the server.

- Command line:

```
jagtool -h eclipse -n 9005 -u admin@system -p jagpass restart
```

- Ant build file:

```
<jag_connect host="eclipse" port="9005" user="admin@system"
password="jagpass" />
<jag_restart />
```

All jagant commands depend on jag_connect. See "Using the jag_connect command" on page 60 for more information about jag_connect.

See also

shutdown

set_props

Description

Sets properties for an entity in the repository. Properties can be set by specifying either a names and values or a properties file.

Syntax

Local-mode support: Yes.

Command line:

set_props [*connect-args* | *local-args*] *entity* [*name value*] | [*file*]

Ant build file, specifying a property file to read:

```
<jag_set_props entity=”entity” file=”file” />
```

Ant build file, specifying properties directly:

```
<jag_set_props entity=”entity” />
  <property name=”name” value=”value”>
  ...
</jag_set_props>
```

Option	Description	Required
connect-args | *local-args*	Arguments to specify a connection to the server or to run in local mode. See “Local versus connected mode” on page 52.	Yes
entity	The entity identifier for the entity whose properties are being set.	Yes
name	The property name. In an Ant build file, you may specify multiple properties as `<property>` elements.	When setting properties directly
value	The property value.	When setting properties directly
file	The name of a property file. Files must specify properties in the format of an EAServer properties file.	When setting properties using a properties file

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

Example 1 This example sets the description for the entity Package:SVU.

- Command line:

```
jagtool set_props Package:SVU com.sybase.jaguar.description “This is the
SVU Package”
```

- Ant build file:

```
<jag_set_props entity="Package:SVU">
<property name="com.sybase.jaguar.description" value="This is the SVU
Package" />
</jag_set_props>
```

Example 2 This example sets the properties for Package:SVU from the file *SVU.props*.

- Command line:

```
jagtool set_props Package:SVU SVU.props
```

- Ant build file:

```
<jag_set_props entity="Package:SVU" file="SVU.props" />
```

Example 3 You can use the Ant build file to specify multiple properties. For example, this declaration sets the values for the com.sybase.jaguar.description and com.sybase.jaguar.package.roles properties for Package:SVU.

- Ant build file:

```
<jag_set_props entity="Package:SVU" />
<property name="com.sybase.jaguar.description" value="This is the SVU
Package" />
<property name="com.sybase.jaguar.package.roles"
value="admin@system,role1" />
</jag_set_props>
```

Example 4 This example sets the host, port, and network protocol values for “MyListener” in the “Jaguar” server:

- Command line:

```
jagtool set_props Listener:Jaguar/MyListener
com.sybase.jaguar.listener.host victor
jagtool set_props Listener:Jaguar/MyListener
com.sybase.jaguar.listener.port 9050
jagtool set_props Listener:Jaguar/MyListener
com.sybase.jaguar.listener.protocol iiop
```

- Ant build file:

```
<jag_set_props entity="Listener:Jaguar/MyListener" />
   <property name=”com.sybase.jaguar.listener.host” value=”victor”>
   <property name=”com.sybase.jaguar.listener.port” value=”9050”>
```

```
        <property name=”com.sybase.jaguar.listener.protocol” value=”iiop”>
    </jag_set_props>
```

Example 5 This jagtool example shows how special characters can be escaped when running commands in DOS or Windows. In this case, the = in the value set must be escaped by quoting:

```
jagtool set_props WebApplication:onepage
com.sybase.jaguar.webapplication.session-config=(session-timeout"="30)
```

This syntax is equivalent:

```
jagtool set_props WebApplication:onepage
com.sybase.jaguar.webapplication.session-config="(session-timeout=30)"
```

See also

Appendix B, “Repository Properties Reference”

shutdown

Description

Shuts down the server to which you are connected.

Syntax

Local-mode support: No.

Command line:

shutdown *connect-args*

Where *connect-args* is a list of arguments to connect to the server, as described in "Using connected mode" on page 52.

Ant build file:

<jag_shutdown />

Like all commands, shutdown requires connection flags at the command line and the jag_connect command in Ant build files (see "Using the jag_connect command" on page 60). shutdown terminates the server process to which you have connected.

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

This example connects to the server *eclipse* on port *9005*, with the user name *admin@system* and the password *jagpass*, and shuts down the server.

- Command line:

```
jagtool -h eclipse -n 9005 -u admin@system -p jagpass shutdown
```

- Ant build file:

```
<jag_connect host="eclipse" port="9005" user="admin@system"
password="jagpass" />
<jag_shutdown />
```

See also

restart

sync

Description

Synchronizes entities in the current repository to one or more remote repositories. Synchronization can be used to create identically configured servers in a cluster, or to copy entities from one server to another.

Syntax

Local-mode support: No.

Command line:

sync *connect-args*
[-clusterfiles *true*|*false*]
[-packagefiles *true*|*false*]
[-servletfiles *true*|*false*]
[-webappfiles *true*|*false*]
[-appfiles *true*|*false*]
[-clientappfiles *true*|*false*]
[-connectorfiles *true*|*false*]
[-syncwebappjavaclasses *true*|*false*]
[-newprimary *true*|*false*]
[-newversion *true*|*false*]
[-refresh *true*|*false*]
[-refresh *true*|*false*]
[-restart *true*|*false*]
[-waitfor *waittime*]
[-verbose *true*|*false*]
[-cluster *clustername*]
[-servers *serverURLS*] *entity*

Ant build file:

<jag_sync
[clusterfiles="*true*|*false*="]
[packagefiles="*true*|*false*="]
[servletfiles="*true*|*false*="]
[webappfiles="*true*|*false*="]
[appfiles="*true*|*false*"]
[clientappfiles="*true*|*false*"]
[connectorfiles="*true*|*false*"]
[syncwebappjavaclasses="*true*|*false*"]
[newprimary="*true*|*false*"]
[newversion="*true*|*false*"]
[refresh="*true*|*false*"]
[refresh="*true*|*false*"]

```
[restart="true|false"]
[waitfor="waittime"]
[verbose="true|false"]
[cluster="clustername"]
[servers="serverURLS"]
entity="entity" />
```

Option	Description	Default	Required
connect-args	Arguments to specify a connection to the server. See "Using connected mode" on page 52.	-	Yes
`clusterfiles`	Indicates all cluster files should be synchronized (all files in Repository/Security, the property file for the cluster, and any server properties found in the cluster definition).	false	No
`packagefiles`	Indicates all packages should be synchronized regardless of the type of entity specified by the *entity* parameter.	false	No
`servletfiles`	Indicates all servlets should be synchronized regardless of the type of entity specified by the *entity* parameter.	false	No
`webappfiles`	Indicates all Web applications should be synchronized, regardless of the type of entity specified by the *entity* parameter.	false	No
`appfiles`	Indicates all applications should be synchronized regardless of the type of entity specified by the *entity* parameter.	false	No
`clientappfiles`	Indicates all application clients should be synchronized regardless of the type of entity specified by the *entity* parameter.	false	No
`connectorfiles`	Indicates all connectors should be synchronized, regardless of the type of entity specified by the *entity* parameter.	false	No
`syncwebappjavaclasses`	When synchronizing Web application files, whether to include class files included in the Web application's custom class list.	true	No
`newprimary`	Indicates that the sync primary specified with this option overrides the default. This option is required when the sync command is initiated from a different server than the one used previously to connect to the current target.	false	See description
`newversion`	Indicates this sync command should result in a new cluster version number. This option is relevant only if the `cluster` option is also specified.	false	No

Option	Description	Default	Required
`refresh`	Refreshes remote objects after the sync command finishes.	false	No
`restart`	Restarts remote servers after the sync command finishes.	false	No
waittime	Indicates a period of time in seconds to wait for a remote server to restart before restarting the next server. This command is relevant only if the `restart` option is also specified.	0	No
`verbose`	Indicates a verbose server output log.	false	No
clustername	The name of the cluster to be synchronized. Either the `cluster` or the `servers` option must be specified.	-	See description
serverURLs	Server URLs in a comma-delimited list. For example: `serverURL1,serverURL2,serverURL3` Either the `cluster` or the `servers` option must be specified.	-	See description
entity	The name of the entity to be synchronized, in the form *EntityType*:*EntityName* as described in "Entity identifiers" on page 54. EntityType must be one of the following: • Cluster • Server • WebApplication • Application • Connector • Package • Component	-	Yes

Return value

Return value	Indicates
0	The command ran successfully; the result is true/success.
2	The command did not run successfully; an exception was thrown.

Examples

This example synchronizes the estore application on the servers in the cluster MyCluster.

- Command line:

```
jagtool sync -cluster MyCluster Application:estore
```

- Ant build file:

```
<jag_sync cluster="MyCluster" entity="Application:estore" />
```

Usage

If you specify a cluster (using the `cluster` option) when a previous sync command to the cluster came from a different source/primary, then you must specify the `-newprimary` option. Otherwise, the sync command fails.

See also

Chapter 6, “Clusters and Synchronization” in the *System Administration Guide*.

Index