Web Services Toolkit User's Guide

EAServer

5.0

DOCUMENT ID: DC31727-01-0500-01

LAST REVISED: December 2003

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About This Book

Audience

The audience for this document is anyone responsible for creating, deploying, and managing Web services. Sybase assumes that these professionals have training in Java and XML and component technology.

How to use this book

Create and manage Web services using the various tools, services, and GUIs described in this book, collectively referred to as Web Services Toolkit:

- Chapter 1, “Overview of Web Services in EAServer” – description of the Web Services Toolkit and the various protocols it supports.
- Chapter 2, “Using Sybase Web Services Toolkit—an Eclipse plug-in” – description of the Eclipse development and management environment.
- Chapter 3, “Components, Datatypes, and Type Mappings” – description of the component types supported as Web services, datatypes, and type mappings.
- Chapter 4, “Web Services Administration” – the procedures to develop and manage Web services from Eclipse.
- Chapter 5, “Web Console—Web Services” – the procedures for managing Web services from the Sybase Management console.
- Chapter 6, “Web Console—Registry Services” – the procedures for managing UDDI registries from the Sybase Management console.
- Chapter 7, “The Private UDDI Server” – the procedures to configure and administer the private UDDI server.
- Chapter 8, “Using wstool, wstkeytool, wstant, and wstkeytoolant” – description of how to use the wstool and wstkeytool command line tools.
- Chapter 9, “Developing Web Service Clients” – description of how to develop client applications from the files generated from the WST development tool and from wstool commands.
- Chapter 10, “Using the Web Services Toolkit Samples” – description of the samples included with WST.

- Appendix A, “Migrating 4.x Web Services” – the procedures for migrating Web services created with WST version 4.*x* to this version of WST.

Related documents

Core EAServer documentation The core EAServer documents are available in HTML format in your EAServer software installation, and in PDF and DynaText format on the *Technical Library* CD.

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

The *EAServer Cookbook* contains tutorials and explains how to use the sample applications included with your EAServer software.

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 System Administration Guide* explains how to:

- Start the preconfigured Jaguar server and manage it with the EAServer Manager plug-in for Sybase Central™
- Create, configure, and start new application servers
- Define connection caches
- 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 or the Repository API

The *EAServer Programmer’s Guide* explains how to:

- Create, deploy, and configure components and component-based applications
- Create, deploy, and configure Web applications, Java servlets, and JavaServer Pages
- Use the industry-standard CORBA and Java APIs supported by EAServer

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 using the Security Manager plug-in for Sybase Central
- 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 Performance and Tuning Guide* describes how to tune your server and application settings for best performance.

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

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://www.sybase.com/detail?id=1024509.

Message Bridge for Java™ Message Bridge for Java simplifies the parsing and formatting of structured documents in Java applications. Message Bridge allows you to define structures in XML or other formats, and generates Java classes to parse and build documents and messages that follow the format. The *Message Bridge for Java User's Guide* describes how to use the Message Bridge tools and runtime APIs. This document is included in PDF and DynaText format on your *EAServer 5.0 Technical Library* CD.

Adaptive Server Anywhere documents EAServer includes a limited-license version of Adaptive Server Anywhere for use in running the samples and tutorials included with EAServer. Adaptive Server Anywhere documents are available on the Sybase Web site at http://sybooks.sybase.com/aw.html.

jConnect for JDBC documents EAServer includes the jConnect™ for JDBC™ driver to allow JDBC access to Sybase database servers and gateways. The *Programmer's Reference jConnect for JDBC* is available on the Sybase Web site at http://sybooks.sybase.com/jc.html.

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 jagtool to configure applications rather than EAServer Manager
variable, *package*, or *component*	Italic font indicates: • Program variables, such as *myCounter* • Parts of input text that must be substituted, for example: *Server*.log • 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 EAServer Manager, a command line, or as program text • Example program fragments • Example output fragments

Other sources of information

Use the Sybase Getting Started CD, the Sybase Technical Library CD and the Technical Library 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 Technical Library CD. It is included with your software. To read or print documents on the Getting Started CD you need Adobe Acrobat Reader (downloadable at no charge from the Adobe Web site, using a link provided on the CD).
- The Technical Library CD contains product manuals and is included with your software. The DynaText reader (included on the Technical Library CD) allows you to access technical information about your product in an easy-to-use format.

  Refer to the *Technical Library Installation Guide* in your documentation package for instructions on installing and starting the Technical Library.

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

  To access the Technical Library Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

Accessibility features

EAServer 5.0 has been tested for compliance with U.S. government Section 508 Accessibility requirements. The online help for this product is also provided in HTML, JavaHelp, and Eclipse help formats, which you can navigate using a screen reader.

EAServer Manager supports working without a mouse. For more information, see "Keyboard navigation" in Chapter 2, "Sybase Central Overview," in the *EAServer System Administration Guide*.

The WST plug-in for Eclipse supports accessibility features for those that cannot use a mouse, are visually impaired or have other special needs. For more information, see "Accessibility features" on page 13.

Note You might 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.

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. Enter user name and password information, if prompted (for existing Web accounts) or create a new account (a free service).

3 Select a product.

4 Specify a time frame and click Go.

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

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

Overview of Web Services in EAServer

Web Services Toolkit (WST) is a set of tools that allows you to create and manage Web services in EAServer. The toolkit supports standard Web services protocols; Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Uniform Description, Discovery, and Integration (UDDI), and includes tools for WSDL document creation, client generation, UDDI registration, and SOAP management.

Web services background and standards

Using Web services and EAServer, you can take advantage of SOAP, WSDL, and UDDI. These protocols enable you to use third-party components called Web services, which are invoked from application providers. A Web service contained in EAServer can be invoked remotely over HTTP and HTTPS protocols. The Web service object has methods or end points that provide the business logic of the Web service being invoked. Methods are called using SOAP, and the client calling these methods is said to consume the Web service. WSDL describes the service and can be used in client applications. You can also publish business and service information to a UDDI registry site on the Web and make your Web service available to other users. SOAP provides a platform and language-neutral way to access these services.

With SOAP, WSDL, and UDDI, collaboration between business partners is easier because interfaces between applications become standardized across platforms.

Web services can be embedded in Sybase's Web container environment. Web services supports these standards:

SOAP 1.1

As part of the Web services functionality, the Simple Object Access Protocol (SOAP) servlet in EAServer provides you with a way to make your EAServer components accessible to your customers with minimal firewall constraints, platform dependencies, or complex development implementations involving Distributed Component Object Model (DCOM) or Common Object Request Broker Architecture (CORBA).

SOAP allows applications to communicate using existing Internet technologies (such as HTTP, URLs, SSL, and XML) and the HTTP or HTTPS port. While SOAP does not mandate which transfer protocol to use, it is the combination of SOAP and HTTP that allows you to invoke remote procedures, even through firewalls.

See the SOAP information pages at http://www.w3.org/TR/SOAP for more information.

WSDL 1.1

As communications protocols and message formats are standardized, it becomes increasingly important to describe these communications in some structured way. The Web Services Description Language (WSDL) addresses this need by defining an XML grammar for describing Web services as collections of communication endpoints capable of exchanging messages. WSDL service definitions provide documentation for distributed systems and for automating the details involved in communication between applications.

When you define a Web service in EAServer, the WSDL file can be automatically generated from the information you provide.

The WSDL document describes a component that you want to make available as a Web service, as well as its location. You can also publish the location of a WSDL document to a UDDI registry on the Web.

The Web services GUI allows you to select a UDDI public host site and login. After you log in, you can add business and service data to the UDDI registry. Once you have published information to the registry, each time you log in, the information is retrieved and available for you to review, modify, or delete.

A business partner can invoke a Web service without knowing how to write SOAP messages by using Web services generated client-side files and artifacts (the collection of files on the client-side that handles communication between a client and a Web service. They include the stub class, service definition interface and additional classes), and the WSDL document that describes your Web service.

See the WSDL information pages at http://www.w3.org/TR/WSDL for more information.

JAX-RPC 1.0

Sun's Java API for XML-based Remote Procedure Call (JAX-RPC) is an API for building Web services and clients that use remote procedure calls (RPCs) and XML. It uses technologies defined by the World Wide Web Consortium (W3C): HTTP, SOAP, and WSDL.

Using JAX-RPC, a remote procedure call is represented by an XML-based protocol (SOAP), which defines the structure, rules, and conventions for representing RPCs and responses. These SOAP messages are transmitted over HTTP or HTTPS. The Java API hides the complexity from the application developer, allowing you to focus on creating the Web services that implement business logic, and the client programs that access them.

See the JAX-RPC Web site at http://java.sun.com/xml/jaxrpc for more information.

JAXM 1.0

Java API for XML Messaging (JAXM) enables applications to send and receive document-oriented XML messages using a pure Java API. JAXM implements SOAP 1.1 so that developers can focus on building, sending, receiving, and decomposing messages for their applications instead of programming low-level XML communications routines.

See the JAXM Web site at http://java.sun.com/xml/downloads/jaxm.html for more information.

JAXP 1.1

Java API for XML Processing (JAXP) supports processing of XML documents using DOM, SAX, and XSLT. JAXP enables applications to parse and transform XML documents independent of a particular XML processing implementation, giving developers the flexibility to swap between XML processors without making application code changes.

See the JAXP Web site at http://java.sun.com/xml/jaxp for more information.

UDDI 2.0

The UDDI specification creates a platform-independent, open framework for describing services, discovering businesses, and integrating business services using the Internet. UDDI is a cross-industry effort driven by major platform and software providers, as well as by marketplace operators and e-business leaders.

Using Web services in EAServer, you can publish a WSDL document that describes your Web service and its location to a UDDI registry.

The UDDI protocol is the building block that businesses can use to transact business with each another, using their preferred applications.

The UDDI specification takes advantage of World Wide Web Consortium (W3C) and Internet Engineering Task Force (IETF) standards, such as eXtensible Markup Language (XML), HTTP, and Domain Name System (DNS) protocols. Additionally, cross-platform programming features are addressed by adopting SOAP.

Web services allows you to publish a WSDL document that describes your Web service and its location to a UDDI registry Web site. A UDDI registry is a sort of yellow pages for businesses, the Web services they offer, and the technical foundations or specifications (called tModels) upon which they are written. You can specify an organization (business name) and description, contact information, and Web service properties for your business. Once your business or tModel is published, potential customers can find it easily from a search. You can publish multiple Web services under the same business name, or create a new business name for different Web services.

Because Web services connect directly to UDDI registry host sites on the Web, you must first be a registered user on the site where you want to publish. To register, go to *www.UDDI.org/register.html*. The UDDI.org Web site maintains a current list of links to UDDI registry host sites where you can register.

EAServer Web Services architecture

Sybase Web Services Toolkit consists of these components:

- The basic SOAP engine, which implements SOAP 1.1, embedded in EAServer.
- The tools for creating and managing Web services:
  - Web-based console for administration, monitoring, and deployment of Web services.
  - Web-based console for UDDI administration, publish/unpublish, and browsing UDDI registries.
  - An Eclipse plug-in GUI that you can use to:
    - Design, develop, and deploy Web services to the EAServer environment.
    - Control deployed Web services running in the EAServer environment.
    - Monitor incoming and outgoing messages for each Web service using a SOAP inspector.
    - Generate standalone Java test clients and JSP clients to invoke Web Services deployed to EAServer environment.
    - Publish and query Web services to or from UDDI registries.

    - Command line tools for designing, developing, deploying, managing, and securing Web services.
- A private UDDI server installed as a J2EE Web application. Access control enables the UDDI user to control access to these basic UDDI data structures: businessesEntity, businessService, bindingTemplate and tModel.

These technologies and tools are collectively referred to as the Web Services Toolkit (WST).

Installing Web services

Web Services is installed as part of a standard EAServer installation. If you customize your installation, you will notice that Web services support consists of:

- WST Runtime – the basic SOAP engine and Web services infrastructure.
- Administration Console – a Web based application described in Chapter 5, “Web Console—Web Services” and Chapter 6, “Web Console—Registry Services.”
- Eclipse based Development Tool – described in Chapter 2, “Using Sybase Web Services Toolkit—an Eclipse plug-in” and Chapter 4, “Web Services Administration.”
- Private UDDI Server – described in Chapter 7, “The Private UDDI Server.”

Defining, deploying, and exposing Web services using WST

WST provides a number of options for defining a Web service, including:

- Importing from a JAR or WAR file – See “Importing a Web service collection” on page 32 and deploy (3) on page 123.
- Creating a Web service from a local or remote WSDL file or Java file – See “Creating Web services from files” on page 34 and
- Creating a Web service from a JAR file – See deploy (2) on page 121.
- Exposing an installed EAServer component as a Web service – See “Other components” on page 57.

- Creating and deploying a Web service from an implementation class file – See deploy (1) on page 120.

Service styles

WST supports the following service styles:

- RPC – the body of the SOAP message is an RPC call containing the method name and serialized versions of the parameters. RPC services use the SOAP RPC conventions, and also encoding rules defined in section five of the SOAP specification.
- Document –the body of the SOAP message is viewed as an XML document, as opposed to an RPC call. Document services do not use any encoding, but still provide XML-to-Java databindings.
- Wrapped – similar to document services, except that rather than binding the entire SOAP body into one big structure, they "unwrap" the body into individual parameters.

Retrieving the Web service's WSDL

To retrieve any WSDL file for a deployed Web service from a Web browser enter the URL of the WSDL in the form http://host:port/collectionName/services/service?wsdl. For example for the canine shelter sample, enter:

http://hostname:8080/SoapSample/services/SoapDemo_FindDog?wsdl.

CHAPTER 2

Using Sybase Web Services Toolkit—an Eclipse plug-in

Eclipse is a full-featured open source software development platform. A Sybase Web Services plug-in to Eclipse provides developers and administrators the ability to manage Web services contained in EAServer. Throughout this book, Eclipse and the Sybase Web Services plug-in together are referred to as the Web Services Toolkit development tool (WST development tool).

The WST development tool provides graphical administration facilities for Web services, including support for development, deployment, and runtime monitoring of Web service-related statistics and messages.

You can develop Web services and create test clients for third-party Web services. However, you can deploy Web services to the runtime engine (EAServer, for example) and create test clients for Web services deployed to EAServer only if you are connected to a running server.

For complete information about Eclipse, see the Eclipse Web site at http://www.eclipse.org.

Starting and stopping Eclipse

You do not need authentication information to start or use Eclipse, but you do need authentication information to connect to a runtime engine in the Web services view of Eclipse. Authentication to EAServer requires the same information from Eclipse as you would supply in EAServer Manager (user name and password).

Note Eclipse is installed as part of the standard EAServer installation. To run Eclipse you must have a complete JDK installation (jdk1.4 or higher), which is not installed as part of the standard EAServer installation.

❖ **Starting Eclipse in UNIX**

- From the command line in the *Shared/eclipse* subdirectory, enter the command:

```
./starteclipse.sh
```

❖ **Starting Eclipse in Windows**

- From the command line in the *Shared\eclipse* subdirectory, enter the command:

```
starteclipse.bat
```

❖ **Stopping Eclipse**

- From Eclipse, select File | Exit

Web services plug-in

The Web services plug-in runs within Eclipse. It is installed when you select the Web Services Toolkit option during the EAServer installation. You can use the WST development tool to define and deploy Web services in projects and applications so that clients can locate and run Web services.

❖ **Accessing Sybase Web Services**

1 Start Eclipse if it is not already running.

2 From Eclipse, select Window | Open Perspective | Other

3 Select Sybase Web Services from the Select Perspective window and click OK.

Connecting to servers

You can manage Web services for any server to which you are connected. See “Web services server administration” on page 30 for more information.

Organization

Sybase Web services contains the following basic units and folders:

- Server – an EAServer runtime process that includes the server name and version, host name on which it is running, and port number to which the WST development tool is connected.
- Web Services – contains the various Web service collections.
- Collection – a group of Web services bundled into a single unit for easy development and management. A collection in a Web services runtime engine is analogous to a Web application in a J2EE container.
- Service – defines the component (EJB, CORBA, Java, PowerBuilder, and so on) that is installed as a Web service. Some aspects of the Web service that you can define include:
  - Ports – the path, URL, or endpoint from which the Web service is made available.
  - Operations – the methods and parameters of the Web service that execute business logic and access data sources.
  - Type Mappings – the name and encoding style of the datatype mapping used by the Web service, depending on the service type (EJB, CORBA, PowerBuilder, and so on).
  - Handlers – contain special routines that can be implemented should a particular event occur. For example, to invoke customized authentication logic, you can write a handler and install it in the Handlers folder.

    - Roles – EAServer's authorization model is based on roles. The roles that are attached to a Web service controls access to that Web service.
- Other Components – contains the packages (a collection of components organized into cohesive, secure units) that are hosted on the EAServer to which the WST development tool is connected. These components can be deployed as Web services if they meet the criteria described in Chapter 3, "Components, Datatypes, and Type Mappings."

Error logging and debugging

Error logging, debugging, and troubleshooting tools consists of several views: Console, Tasks, SOAP Inspector, and Web Services Console. From the WST development tool, select Window | Show View | and:

- Console – displays the output of the execution of programs and allows you to enter input for the program. The console shows three different kinds of text, each in a different color:
    - Standard output
    - Standard error
    - Standard input
- Web Services Console – displays the messages, errors, and warnings generated whenever you perform a Sybase Web services action. The Web services console allows you to monitor the various log files; *Jagaur.log*, *Jaguarhttpservletl.log*, *Jaguarhttprequest.log*, and so on, by selecting the file from the Log file drop-down list.
- Tasks – displays auto-generated errors, warnings, or information associated with a resource. Double-click an item in the Task view to display more detailed information.
- SOAP Inspector – displays incoming and outgoing messages for a given Web service. Each Web service displays in an Inbound Messages folder and an Outbound Messages folder that includes the protocol, name of the host, port number where the Web service is made available, and the name of the Web service. Double-click the Web service to view either outbound or inbound traffic. The SOAP or HTTP responses, which depend on the tab you select, appear in the right pane.

There is also an Eclipse log, *${eclipse.home}\workspace\.metadata\.log* ,where errors are logged, as well as the EAServer log file, located in the *bin* subdirectory of your EAServer installation.

Menu layout and navigation

The WST development tool provides panes and tabs that provide views of Web service-related properties and resources.

From the WST development tool, select Window | Show View | and:

- Sybase Web Services – the Web services, properties, and resources for the server to which the WST development tool is attached. Perform most Web service administrative tasks from this pane as described in Chapter 4, “Web Services Administration.”
- Package Explorer – the contents of the projects, plug-ins, JAR files, and so on for Web service projects and packages. View the contents of a file by right-clicking a file and selecting Open (or Open Hiearchy). The selected file displays in the right pane.

Accessibility features

WST supports accessibility features for those that cannot use a mouse, are visually impaired or have other special needs. For information about these features refer to 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

CHAPTER 3

Components, Datatypes, and Type Mappings

Using WST, you can create a Web service from an EAServer component and use SOAP to expose it across your firewall. You can select any components in EAServer for a Web service that have return values or parameters of supported datatypes. The components you select for a Web service must be installed in EAServer.

Web services use XML to transfer data between service endpoints. WST includes standard mappings for some basic Java datatypes to XML and vice versa. It also allows you to create user-defined datatypes and mappings for complex datatypes.

Supported component types

WST supports the following component types as Web services:

- Stateless EJBs
- Stateless Java–CORBA
- Stateless C++–CORBA
- Stateless PowerBuilder
- Class files

Note Supported components must contain supported datatypes, including user-defined datatypes to be a valid Web service. See the *EAServer Programmer's Guide* for information about stateless components.

Supported datatypes

This section describes the datatypes supported in WST. The datatype must belong to a supported component type for it to be available as a Web service. Supported datatypes include:

- JAX-RPC defined data types – Refer to chapter four (WSDL/XML to Java Mapping) and five (Java to XML/WSDL Mapping) of the *Java API for XML-based RPC JAX-RPC 1.0* specification. See the JAX-RPC download site at http://java.sun.com/xml/downloads/jaxrpc.html
- Java with IDL datatypes – the component's method declarations use the datatype mappings that are specified by the CORBA document, *IDL to Java Language Mapping Specification* (formal/99-07-53).
- CORBA C++ with IDL datatypes – the component's method declarations use the OMG standard for translating CORBA IDL to C++. For more specifics, see *C++ Language Mapping Specification* (formal/99-07-41). You can download this document from the OMG Web site at http://www.omg.org. C++ datatype mappings are the same as the Java/IDL component datatype mappings that are listed in Table 3-1.

Table 3-1 lists the datatypes supported in WST and EAServer by default, and the equivalent XML XSD types.

Table 3-1: Java datatype and XML equivalents

XML XSD type	Java datatypes
xsd:boolean	org.omg.CORBA.BooleanHolder
xsd:byte	org.omg.CORBA.ByteHolder
xsd:double	org.omg.CORBA.DoubleHolder
xsd:float	org.omg.CORBA.FloatHolder
xsd:int	org.omg.CORBA.IntHolder
xsd:long	org.omg.CORBA.LongHolder
xsd:short	org.omg.CORBA.ShortHolder
xsd:string	org.omg.CORBA.StringHolder
xsd:byte	BCD.BinaryHolder
xsd:decimal	BCD.Decimal
xsd:decimal	BCD.DecimalHolder
xsd:base64Binary (same as byte[])	BCD.Money
xsd:base64Binary (same as byte[])	BCD.MoneyHolder
xsd:double	MJD.Date
xsd:double	MJD.Time
xsd:double	MJD.Timestamp

Additional datatype support

In addition to the datatypes described in Table 3-1, Web services supports java.sql.ResultSet and TabularResults.ResultSet, which maps to a complex schema element that contains the resultset data and the schema for the resultset:

For java.sql.ResultSet:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="jdbc.wst.sybase.com">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
   <complexType name="DataReturn">
     <sequence>
         <element name="XML" nillable="true" type="xsd:string" />
         <element name="updateCount" type="xsd:int" />
         <element name="DTD" nillable="true" type="xsd:string" />
         <element name="schema" nillable="true" type="xsd:string" />
     </sequence>
  </complexType>
</schema>
</wsdl:types>
```

For TabularResults.ResultSet:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="TabularResults.wst.sybase.com">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
   <complexType name="DataReturn">
      <sequence>
         <element name="XML" nillable="true" type="xsd:string" />
         <element name="updateCount" type="xsd:int" />
         <element name="DTD" nillable="true" type="xsd:string" />
         <element name="schema" nillable="true" type="xsd:string" />
      </sequence>
   </complexType>
</schema>
</wsdl:types>
```

Client-side generation of holder classes

When you expose a component that uses EAServer-specific holder types as a Web service, the convention for generating the client-side holders classes is that they are always generated under a package.holders.type hierarchy. For example, when you expose a component as a Web service that uses holder type BCD.MoneyHolder, the conversion on the client-side results in a JAX-RPC specific holder contained under BCD.holders.MoneyHolder. You cannot use EAServer specific types on the Web service client side.

Custom datatypes and mappings

Theoretically, any object or datatype that can be described by a valid snippet of XML can be used as a parameter within a SOAP call. Current support of advanced datatypes in Web services is based on the serialization framework specified by the JAX-RPC 1.0/1.1 specification. See the JAX-RPC Web site at http://java.sun.com/xml/jaxrpc for more information.

Creating custom type mappings

This section describes how to use the WST development tool to create type mappings and associate a Web service with a type mapping.

❖ **Creating a Web service type mapping**

1 Select File | New | Other | Sybase Web Services | Type Mapping.

2 Follow the wizard instructions to create the type mapping. Table 3-2 describes the Web service type mapping properties.

Table 3-2: Web service type mapping wizard

Window	Property	Description
Type Mapping Classes Selection	Type Class	A user defined structure or Java class whose mapping to XML is not standard. For example java.sql or resultsets.
	Serializer Class	The fully qualified name of the serialization class used to convert the new datatype to XML.
	Deserializer Class	The fully qualified name of the deserialization class used to convert the serialized XML data into application data.
	SerializerFactory Class	An instance of the serialization class.
	DeserializerFactory Class	An instance of the deserializer class.
Type Mapping WSDL Definition	Use Web Service Target NameSpace as Type Mapping's NameSpace	Each type mapping can have its own namespace or have the same namespace as the Web service's target namespace. If this property is selected, then "Type Mapping Namespace" is disabled.
	Type Mapping NameSpace	The type mapping namespace (if not using the Web service's target namespace).
	Local Part	The local part of a qualified name (QName) which consists of a namespace plus ":" plus a local part serves as a pointer to a WSDL definition part.
	Encoding Style	The encoding style used by the XML parser to apply when transforming a SOAP message to a Java object. Use "SOAP" unless you have defined an alternative encoding style for this class.
Store the Created Type Mapping to Local Store	Select Local Store or Create New Store	You can select an existing store for this type mapping or create a new one. Normally, a type mapping store consists of a description file and a list of JAR files. The description file contains the information of the first two windows, the JAR contains the serializer, deserializer, serializer factory, and deserializer factory classes. You can select "Sybase Web Service View," and select type mappings to import them into the desired local store.

Window	Property	Description
Undefined Type Mapping Found in Class	Please Define the Undefined Type Mapping Found for This	When you click "Select a Java file," and create a Web service from it, this wizard displays if the selected Java file contains an undefined type mapping. Click Add to launch the Web Service Type Mapping creation wizard to create a type mapping for this datatype.

Java coding standards
Web Services Toolkit follows Java coding standards. When you use any Java class name in your Web service, or user defined types in the IDL, the name must start with an upper case letter. If the names or types start with a lower case letter, you might see a "class not found" error.

Creating serializers and deserializers

There might be instances where the existing serializers and deserializers provided with the WST are not adequate to expose a class or component through SOAP. In this case, you must create custom serializer and deserializer classes to perform the necessary actions to convert the class to and from XML.

A new serializer and deserializer requires a new Java class that implements the `javax.xml.rpc.encoding.Serializer` for the serializer and `javax.xml.rpc.encoding.DeSerializer` for the deserializer.

The following `nonbeansample` example illustrates various aspects of creating a serializer and deserializer for a user-defined datatype.

Description

The following listing contains these files:

- Book.java – the type class, which needs a custom serializer/deserializer since it's not a valid Java Bean or a type for which WST provides built in mappings (like IDL types).
- BookSerFactory – the factory used to get the serializer. Currently WST supports only SAX serializer/deserializer, but factory is the interface to get XML parser specific serializers/deserializers.
- BookDeserFactory – the factory used to get the deserializer.
- BookSerializer – contains the logic to convert Java type to XML, also contains write schema which can be implemented. Write schema is used during WSDL generation. This class implements `javax.xml.rpc.encoding.Serializer`.

- BookDeserializer – contains the logic to convert XML to java type. This class is an extension of `org.apache.axis.encoding.DeserializerImpl` and provides the base functionality. The `deserializerImpl` class implements the `javax.xml.rpc.encoding.Deserializer`. You can also write all the deserialization logic on your own.

Listing

```
 /*
*/
package nonbeansample;

import org.apache.axis.encoding.DeserializerImpl;
import org.apache.axis.Constants;
import org.apache.axis.encoding.DeserializationContext;
import org.apache.axis.encoding.Deserializer;
import org.apache.axis.encoding.FieldTarget;
import org.apache.axis.message.SOAPHandler;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;

import javax.xml.namespace.QName;
import java.util.Hashtable;

/**
 *
 *
 */
public class BookDeserializer extends DeserializerImpl {
        public static final String NAMEMEMBER = "name";
        public static final String AUTHORMEMBER = "author";
        public static final QName myTypeQName = new QName("typeNS", "Book");

        private Hashtable typesByMemberName = new Hashtable();

        public BookDeserializer()
        {
               typesByMemberName.put(NAMEMEMBER, Constants.XSD_STRING);
               typesByMemberName.put(AUTHORMEMBER, Constants.XSD_STRING);
               value = new Book("","");
        }

        /** DESERIALIZER - event handlers
         */

        /**
```

```
         * This method is invoked when an element start tag is encountered.
         * @param namespace is the namespace of the element
         * @param localName is the name of the element
         * @param prefix is the element's prefix
         * @param attributes on the element...used to get the type
         * @param context is the DeserializationContext
         */
        public SOAPHandler onStartChild(String namespace,
                                        String localName,
                                        String prefix,
                                        Attributes attributes,
                                        DeserializationContext context)
               throws SAXException
        {
        QName typeQName = (QName)typesByMemberName.get(localName);
        if (typeQName == null)
        throw new SAXException("Invalid element in Book struct - " +
localName);

               // These can come in either order.
               Deserializer dSer =
context.getDeserializerForType(typeQName);
               try {
               dSer.registerValueTarget(new FieldTarget(value, localName));
               } catch (NoSuchFieldException e) {
                      throw new SAXException(e);
               }

               if (dSer == null)
                      throw new SAXException("No deserializer for a " +
typeQName + "???");

               return (SOAPHandler)dSer;
        }

}
/*
 *
 *
 * package nonbeansample;

import org.apache.axis.encoding.DeserializerFactory;

import org.apache.axis.Constants;
import java.util.Iterator;
import java.util.Vector;
```

```
/**
 * *
 *
 */
public class BookDeserFactory implements DeserializerFactory {
        private Vector mechanisms;

        public BookDeserFactory() {
        }
        public javax.xml.rpc.encoding.Deserializer getDeserializerAs(String
mechanismType) {
               return new BookDeserializer();
        }
        public Iterator getSupportedMechanismTypes() {
               if (mechanisms == null) {
                      mechanisms = new Vector();
                      mechanisms.add(Constants.AXIS_SAX);
               }
               return mechanisms.iterator();
        }
}
/*
 *
 *
 * */
package nonbeansample;


/**
 *
 *
 * */
public class Book {
    /** book name */
    public String name;

    /** book author */
    public String author;

    /**
     * Constructor.
     * @param name book name
     * @param author book author
     * @throws IllegalArgumentException name or author is null
     */
```

```
    public Book(String name, String author) {
        if (name == null) {
            throw new IllegalArgumentException("Name is null!");
        }

        if (author == null) {
            throw new IllegalArgumentException("Author is null!");
        }

        this.name = name;
        this.author = author;
    }

    /**
     * Test for equality.
     * @param object any object
     * @return true if books are equal
     */
    public boolean equals(Object object) {
        if (!(object instanceof Book)) {
            return false;
        }

        Book secondBook = (Book) object;

        return name.equals(secondBook.name) &&
        author.equals(secondBook.author);
    }
}
/*
 * */
package nonbeansample;

import java.util.Iterator;
import java.util.Vector;
import org.apache.axis.Constants;

import org.apache.axis.encoding.SerializerFactory;

/**
*/
public class BookSerFactory implements SerializerFactory {
        private Vector mechanisms;

        public BookSerFactory() {
        }
```

```
        public javax.xml.rpc.encoding.Serializer getSerializerAs(String
mechanismType) {
               return new BookSerializer();
        }
        public Iterator getSupportedMechanismTypes() {
               if (mechanisms == null) {
                      mechanisms = new Vector();
                      mechanisms.add(Constants.AXIS_SAX);
               }
               return mechanisms.iterator();
        }
}
*/
package nonbeansample;

import java.io.IOException;

import javax.xml.namespace.QName;

import org.apache.axis.encoding.SerializationContext;
import org.apache.axis.encoding.Serializer;
import org.apache.axis.wsdl.fromJava.Types;
import org.w3c.dom.Element;
import org.xml.sax.Attributes;
import org.apache.axis.Constants;

/**
*/
public class BookSerializer implements Serializer {
        public static final String NAMEMEMBER = "name";
        public static final String AUTHORMEMBER = "author";
        public static final QName myTypeQName = new QName("nonBeanTypes",
"Book");

        /** SERIALIZER
         */
        /**
         * Serialize an element named name, with the indicated attributes
         * and value.
         * @param name is the element name
         * @param attributes are the attributes...serialize is free to add
more.
         * @param value is the value
         * @param context is the SerializationContext
         */
        public void serialize(
```

```
                QName name,
                Attributes attributes,
                Object value,
                SerializationContext context)
                throws IOException {
                if (!(value instanceof Book))
                       throw new IOException(
                              "Can't serialize a "
                                     + value.getClass().getName()
                                     + " with a BookSerializer.");
                Book data = (Book) value;

                context.startElement(name, attributes);
                context.serialize(new QName("", NAMEMEMBER), null,
data.name);
                context.serialize(new QName("", AUTHORMEMBER), null,
data.author);
                context.endElement();
        }
        public String getMechanismType() {
                return Constants.AXIS_SAX;
        }

        /* (non-Javadoc)
         * @see
org.apache.axis.encoding.Serializer#writeSchema(java.lang.Class,
org.apache.axis.wsdl.fromJava.Types)
         */
       public Element writeSchema(Class arg0, Types types) throws Exception
{
                //  Auto-generated method stub
                Element complexType = types.createElement("complexType");
                types.writeSchemaElement(myTypeQName, complexType);
                complexType.setAttribute("name",
myTypeQName.getLocalPart());
                Element seq = types.createElement("sequence");
                complexType.appendChild(seq);

                Element element = types.createElement("element");
                element.setAttribute("name", "name");
                element.setAttribute("type", "xsd:string");
                seq.appendChild(element);
                Element element2 = types.createElement("element");
                element2.setAttribute("name", "author");
                element2.setAttribute("type", "xsd:string");
                seq.appendChild(element2);
```

```
                return complexType;
            }
    }
```

Deployment

The deployment of custom type mappings requires serializer and deserializer classes and the associated factories, the SOAP encoding, and the qualifying name (Qname). The Qname helps reduce the chance of collisions of elements that use the same name (description, item, and other entities), by adding an additional element, which makes it more likely to produce a unique element:

```
Qname = namespace identifier + local name
```

The deployment of type mappings occurs as part of Web service creation and deployment. Type mappings are deployed at the Web service level.

Scope of type mappings

A type mapping can be deployed only as part of a Web service, or when exposing a component. Even though a type mapping is deployed as only part of a service, a client can look up all the service mappings installed on a server.

Once a type mapping is deployed, it is associated with a Web service. But the same type mapping can be used with other Web services as well.

Exporting

From the Web service Server view, you can select a Type Mapping and export. You can then specify the local store, and import the JAR and additional information (Type Mapping Namespace, Local Part, and Encoding Style) to the selected local store.

CHAPTER 4

Web Services Administration

This chapter describes how to administer Web services from the WST development tool.

You can perform many of the same functions described in this chapter using wstool and wstkeytool commands. See Chapter 8, “Using wstool, wstkeytool, wstant, and wstkeytoolant” for more information.

Introduction

The WST development tool supports top-down (creating a Web service from a component) and bottom-up (creating a Web service from the WSDL) development of Web services, deployment of Web services to the runtime engine, and UDDI publication and unpublication.

You can manage certain aspects of the Web service container, create and manage Web service projects, and troubleshoot Web services using logs and the SOAP inspector.

Before you can manage Web services, you must install the Web service plug-in. See Chapter 2, “Using Sybase Web Services Toolkit—an Eclipse plug-in” for more information.

Web services server administration

A Web services server is the container on EAServer that stores your Web services. You can create any number of server profiles that allow you to connect to a Web services container and manage the Web services that it contains.

Note When managing Web services, the server must be running. You can develop Web services and create test clients for third-party Web services without connecting to the server.

❖ **Creating and modifying a Web services server profile**

1 Right-click the Sybase Web Services Servers icon and select Create Server profile.

2 The Create Server Profile dialog box appears. Provide the information described in Table 4-1 and click Finish. If a profile already exists, you can select the profile, make modifications and click Finish.

Table 4-1: Create server profile properties

Property	Description
Profile Name	The name of the Web services server profile you are creating.
User Name	The name of the user connecting to the Web services container. `jagadmin` is the default. Use either `jagadmin` or another member of the Admin role.
Password	The password of the user connecting to the Web services container. The default is blank.
Host Name	The name of the host machine that contains the Web services container to which you are connecting. `localhost` is the default.
Port Number	The port number of the host used to connect to the Web services container. `8080` is the default.
Server Startup Script File (optional)	This path to the script if you providing connection information in a script.
Script Arguments	Any additional arguments you want to provide to the script.

❖ **Setting the default Web services server**

If you have multiple Web services servers, you can designate a default to which you connect when you start the WST development tool.

1 Right-click the server profile you are designating as the default.

2 Select Set Default.

❖ **Connecting to a Web services server**

You must be connected to a Web services server to manage Web service collections, Web services, and so on. If you cannot connect to the server, make sure it is running.

- Right-click the server profile and then select Connect.

❖ **Disconnecting from a Web services server**

- Right-click the server profile and then select Disconnect. Only available if you are connected to the server.

❖ **Starting a Web services server**

1 Right-click the server profile to which the Web services server you are starting belongs.

2 Select Start.

❖ **Stopping a Web services server**

1 Right-click the server profile to which the Web services server you are stopping belongs.

2 Select Stop.

❖ **Refreshing a Web services server**

You must start a Web services server before refreshing.

1 Right-click the server profile to which the Web services server you are refreshing belongs.

2 Select Refresh.

❖ **Restarting a Web services server**

1 Right-click the server profile to which the Web services server you are restarting belongs.

2 Select Restart.

❖ **Removing a Web services server**

1 Right-click the server profile to which the Web services server you are removing belongs.

2 Select Remove.

Web services collection administration

A Web services collection is a logical group of Web services contained in a folder. You can manage collections only for the Web services server to which you are connected. When you deploy a Web service to a server, it is placed in a Web service collection. The default Web service collection is “ws.”

❖ **Importing a Web service collection**

You can import a Web service collection into the Web services development tool from a WAR file.

1 Right-click the Web services icon and then select Import.

2 Enter, or browse for the Web service collection you are importing.

3 Click OK. The Web service collection is imported.

❖ **Exporting a Web services collection**

You can export a Web service collection and all the Web services it contains to a WAR file.

1 Expand the Web Services icon.

2 Right-click the Web service collection you are exporting and select Export Collection.

3 Enter the file name and location to which you are exporting the Web services collection.

4 Click OK. The Web services collection is exported. You can now import the Web services collection WAR file into other servers.

❖ **Refreshing a Web services collection**

If you make changes to a Web service collection, for example if you deploy a Web service to a Web service collection, refresh the collection so you can see the most current changes.

1 Highlight the server to which the Web service collection belongs.

2 Right-click the Web service collection, then select Refresh.

❖ **Deleting a Web services collection**

1 Highlight the server to which the Web service collection belongs.

2 Right-click the Web service collection and then select Delete.

Table 4-2 describes the Web services collection properties.

Table 4-2: Web service collection properties

Property	Description
Name	The name of the Web services collection.
Description	A description of the Web services collection.

Web service administration

This section describes how to create Web services and add them to a Web service collection, and manage existing Web services. See “Exposing Components as Web services” on page 48 for information about deploying existing components as Web services.

Creating Web services from files

This section describes how to:

- Create a Web service from a WSDL file – this bottom-up approach (creating a Web service from the WSDL) allows you to create a Web service from an existing WSDL file.
- Create a Web service from a Java file – this top-down approach (creating a Web service from a component) allows you to create a Web service from a Java file.

The Web service can be contained in various projects. See “Web service projects” on page 36 for more information about projects.

❖ **Creating a Web service from a WSDL**

1 From the Web Service container, select File | New | Other.

2 The New wizard displays. Select Sybase Web Services in the left pane, and Web Service in the right pane. Click Next. You can also create the Web service within a project by selecting Web Service Project. If you do not select a project at this time, you will be asked later to provide a project for the Web service.

3 The Create Web Service wizard displays. Follow the instructions to create a Web service from a WSDL file. Table 4-3 on page 36 describes the wizard properties.

4 Complete the wizard instructions and click Finish to create the Web service. If you choose a Project for this Web service, you can view the project by selecting Window | Show View | Package Explorer. The Projects appear in the right pane. Expand the project and package to view the Web service. Along with a Web service, the wizard generates the other required files, including a *.wsdd* file.

You can right-click the *.wsdd* file and then select Deploy to deploy it as a Web service.

❖ **Creating a Web service from a Java file**

1 From the Web Service container, select File | New | Other.

2 The New wizard displays. Select Sybase Web Services in the left pane, and Web Service in the right pane. Click Next. You can also create the Web service within a project by selecting Web Service Project. If you do not select a project at this time, you are asked later to provide a project for the Web service.

3 The Create Web Service wizard displays. Follow the instructions to create a Web service from a Java file. Table 4-3 describes the wizard properties.

4 Complete the wizard instructions and click Finish to create the Web service. If you choose a Project for this Web service, you can view the project by selecting Window | Show View | Package Explorer. The Projects appear in the right pane. Expand the project, and package to view the Web service. Along with a Web service, the wizard generates the other required files, including a *.wsdd* file.

You can right-click the *.wsdd* file and then select Deploy to deploy it as a Web service.

Table 4-3: Web service creation wizard options and properties

Window	Property	Description
Select the Web Service Project	Project Type	Select the project in which you will create a Web service. The project wizard displays only if you choose to create a Web service project.
Create the Project	Project Name	Provide a name for your project.
	Project Contents	Use the Browse button to select the project contents directory that contains your project, or click the check box to use the default directory, which is the project name located in the *$Eclipse/workspace* directory.
Select Approach	Create from WSDL or Create from Java File	You can create the The Web service from an existing Java file or *.wsdl* file. Click the appropriate check box.
If Creating From WSDL	Locate From a Local File, URL, or UDDI	Provide the location of the *.wsdl* file, by entering the file location, URL, or UDDI site. If the file is on the local file system use Browse to locate it. If you are locating the file from a UDDI site, follow the instructions for publishing to a UDDI site as described in Table 4-8 on page 55.
	Package Name	The name of the package in which the Web service is created. If you do not enter a package name, “default” is used.
If Creating From Java File	Create From Java File	Enter the Java file being used to create the Web service.
	Options	You can specify various preferences used for you Web service deployment. These options are described in Table 4-6 on page 50.
	Method Selection	Select the methods/operations to be exposed in the Web service’s WSDL file.
Summary		A summary of your entries. Verify they are accurate and click Finish, or Back to change your selections.

Web service projects

The WST development tool allows you to create and maintain various projects that contain collections of Web services, class files, readme files, and so on, that make up a Web service project depending on your need. For example, you can create:

- Server projects – generate and contain the server-side files required to deploy a Web service project to the server.
- Client projects – generate and contain the client-side files required to deploy a Web service project to the client.
- Projects – generate and contain both the server-side and client-side files required to deploy a Web service project to the server and client.

Sybase recommends that when creating projects, you keep the client-side code in a client project and server-side code in a separate server project. This allows you to generate, compile, and maintain the client-side and server-side files, artifacts, and dependent classes independently.

To get an idea of how projects can be used to keep track of your various Web service projects, See "Creating the sample projects and installing the samples" on page 166.

Web service management

This section describes how to use the WST development tool to manage Web services already contained in a server. Each procedure described in this section requires that you first:

1 Connect to the server that contains the Web service.

2 Expand the Web Services icon.

3 Expand the Web service collection to which the Web service belongs.

❖ Viewing the WSDL

1 Right-click the Web service and then select View WSDL.

2 The WSDL file for this Web service displays in the right pane. You cannot edit this file.

❖ Activating a Web service

If a Web service is already activated, this option is dimmed; clients can only access a Web service that is activated:

1 Right-click the Web service and then select Activate.

❖ Deactivating a Web service

If a Web service is already deactivated, this option is dimmed.

- Right-click the Web service and then select Deactivate.

❖ **Refreshing a Web service**

Refresh a Web service if you make any changes to it.

- Right-click the Web service and then select Refresh.

❖ **Deleting a Web service**

1 Right-click the Web service and then select Delete.

Creating and managing Web service clients

This section describes how to create and manage Web service clients from a Web service. Each procedure requires that you first:

1 Connect to the server that contains the Web service.

2 Expand the Web Services icon.

3 Expand the Web service collection to which the Web service belongs.

Note The wizards described in this section generate a test client runtime JAR file, *sybasewstrt.jar,* which contains one file, *manifest.mf*, that lists the JAR files required by the runtime client:

- When compiling the client class, do not include *sybasewstrt.jar*. Set the required JARs in the classpath individually.
- The classpath should include at a minimum: *sybasewstrt.jar*, *sybasewst.jar*, *jaxrpc.jar*, and the path to the client artifacts.
- When running the client, use either the "-classpath" option, or "set classpath" to specify the location of the required files identified by *sybasewstrt.jar*.

After using the wizard to generate the various files required by the client, see Chapter 9, "Developing Web Service Clients" for a description of how to develop a client.

❖ **Creating a Web service client**

1 Right-click the Web service and then select Create Web Service Client.

2 The Create Web Service Client wizard displays.

3 Follow the wizard instructions described in Table 4-4. Click Finish when done.

4 The wizard generates the test client, and necessary client artifacts in the package you specify.

Table 4-4: Create Web service client wizard options and properties

Window	Property	Description
Select a Project	Project Name	The wizard displays a list of available projects. Highlight the project to which the client you are generating belongs.
Java Package	Package	The name of the package where the client is generated. Enter a name of a package, or use the drop down list to locate an existing package.
WSDL2Java Options	Generate Code for this WSDL Only	Select this checkbox to generate code only for this WSDL document. Uncheck (The default) to generate files for all WSDL documents, the immediate one and all imported ones.
	Timeout	The time, in seconds, for this operation to complete successfully before timing out. In case of timeout, check the log files for possible reasons.
	Use Special Treatment for "wrapped" Document/Literal	Allows support for "wrapped" document/literal. Wrapped is a document literal variation, that wraps parameters as children of the root element. Uncheck this box to turn off the special treatment of "wrapped" document/literal style operations. If checked (the default), WSDL2Java recognizes these conditions: • An input message has is a single part • The part is an element • The element has the same name as the operation • The element's complex type has no attributes Under these conditions, the top level elements are "unwrapped", and each component of the element is treated as an argument to the operation. This type of WSDL is the default for Microsoft .NET Web services, which wraps RPC style arguments in this top level schema element.
	Type Mapping Version	The type mapping version. Valid options are 1.1 (the default) and 1.2. This option determines which version of SOAP the Web service uses, SOAP 1.1 or SOAP 1.2.
	Generate Code for All Elements	Allows you to generate and compile the stubs, wsdd, and ImplTemplate files.
	Emit separate helper classes for meta data	Helper classes are used by the primary class to help execute its business methods/operations. Helper classes are normally generated for user defined type beans. You can think of them as wrappers for the user defined beans that contain information (utility methods) which is used at runtime. They allow you to write your own Java beans with custom behavior and use them in the runtime SOAP stack.

Window	Property	Description
	User name	The user name used to access the WSDL URI.
	Password	The password required by the user to access the WSDL URI.
Summary		Contains information from the previous pages. Review and click Finish to accept your selections, or Back to change.

❖ **Creating a JSP client**

This procedure generates JSP client pages from the Web service and stores them on the server. Once created, you can test the JSP pages by Launching the JSP client.

- Right-click the Web service and then select Create JSP client.

❖ **Launching a JSP client**

This procedure launches the JSP client you created in the proceeding procedure, by starting a Web browser, and running the JSP.

- Right-click the Web service and then select Launch JSP Client.

❖ **Deleting a JSP client**

If you created a JSP client for this Web service, this procedure deletes it.

- Right-click the Web service and then select Delete JSP Client.

Web service operation management

This section describes how to manage Web service operations (or methods). These procedures require that you:

1 Expand the Web service collection.

2 Expand the Web service.

3 Expand the operations folder.

Overloaded methods

If you deploy a Web service that contains overloaded methods, the WST development tool displays only the first method of the overloaded method. Allowing or disallowing access to the method, affects all overloaded methods.

For example, if the Web service contains an overloaded method that contains the methods echo(String, String) and echo (String), the GUI displays only echo (String, String) twice, but the allowed/disallowed operation affects both echo(String, String) and echo(String).

❖ **Invoking an operation**

This procedure invokes an operation of the Web service to which it belongs.

- Right-click the operation and then select Invoke.

❖ **Allowing an operation**

Allowing a Web service operation makes it available to clients. If a Web service operation is already allowed, this option is dimmed.

- Right-click the operation and then select Allow.

❖ **Disallowing an operation**

Prevent access to a Web service operation by following this procedure. If a Web service operation is already disallowed, this option is dimmed.

- Right-click the operation and then select Disallow.

Table 4-5 describes the Web service operation properties.

Table 4-5: Web service operation properties

Property type	Property	Description
General	Name	The name of the operation.
	Description	A description of the Web service operation.
	Style	The SOAP binding style: • Document – indicates that the SOAP body contains an XML document, or • RPC (remote procedure call) – indicates that the SOAP body contains an XML representation of a method/operation call.
	Return Type	Specifies the return type of the operation.
	Is return value in response message	True or false.
	SOAP Action	The URI for the SOAPAction HTTP header for the HTTP binding of SOAP. The SOAPAction HTTP request header field can be used to indicate the intent of the SOAP HTTP request. The URI identifies the intent.
	Message Operation Style	Document, RPC, or wrapped.

Type mappings

Type mappings are described in Chapter 3, “Components, Datatypes, and Type Mappings.”

Handlers

A handler is a Java class that implements org.apache.axis.Handler. They are contained in a Handlers folder for each Web service. You can have multiple handlers for each Web service. A handler class can be deployed to the server in the following ways:

- If deploying a service from the WST development tool, include the handler class file in the current Web services project. When the service is deployed to the server, the handler class is also deployed.
- If the handler class is contained in a JAR file, manually copy the JAR to the *java/classes* subdirectory of your EAServer installation, and add the JAR file to the server’s classpath. This may be a more efficient method if the handler is to be used by more than one Web service.

You can define two types of handlers:

- Request handlers – are invoked before the actual Web service method is invoked. For example, you may have a handler that implements customized authentication logic, depending on the request.
- Response handlers – is invoked after the actual Web service method is invoked. For example, you may have a handler that sends the contents of a SOAP message after the method is invoked.

Handlers can not be created, edited, or moved using the WST development tool, only added and deleted.

Defining handlers

A handler definition can be added to a Web service provided the handler class has already been deployed or installed in the server, as described above.

The WST development tool and the Web console have menu options to add a handler from the Handlers folder, and set necessary properties for the handler. A request and response handler contains these properties:

- A name by which the handler is identified
- The Java class that implements the handler
- Any parameters that the handler requires

Some examples of built-in handlers include:

- org.apache.axis.handlers.LogHandler
- org.apache.axis.handlers.SimpleSessionHandler.

Refer to the Java documentation of these handlers for more information. Java documentation can be accessed from the EAServer contents screen at http://host_name:8080 where *host_name* is the name of the your EAServer host. localhost is the default.

Security

This section describes how to implement security for Web services.

Roles and security realms

This section contains the procedures for establishing security for your Web services from the WST development tool. Each procedure described in this section requires that you first connect to the server that contains the Web service.

Establishing Web services security is based on roles. For complete information about roles, see the *EAServer Security Administration and Programming Guide*.

Web services security tutorial

EAServer includes a Web services security tutorial that familiarizes you with establishing different levels of security for a Web service and its methods/operations. See Chapter 10, “Using the Web Services Toolkit Samples” for more information.

Managing security realms and roles

EAServer contains a default security realm. The security realm is a container used to store the roles used to allow, and limit, access to your Web services. When you connect to EAServer from the WST development tool, you see the security realm.

❖ **Refreshing a security realm**

If you add a role to a security realm or make any other changes, refresh the realm.

- Right-click the security realm and then select Refresh

Managing roles

A role can consist of authorized users, authorized digital Ids and authorized operating system users. Create a role in a security realm. Add roles at the Web service or Web service operation level to restrict access to those resources.

Note When you manage roles from the WST development tool, you manipulate the Repository of the server to which you are connected. When you add, delete, or otherwise modify a role, those changes are reflected in EAServer Manager.

❖ **Creating a role**

1 Expand the security realm.

2 Right-click the Roles icon and then select Create role.

3 Enter a role name and description and click OK.

❖ **Deleting an existing role**

1 Expand the security realm.

2 Expand the roles icon.

3 Right-click the role and then select Delete.

❖ **Allowing a user, group, or digital ID access to a role**

Each role can include specific user names and digital IDs. If you use native operation system authentication, you can also include operating system group names; all users in the specified group are affected.

1 Expand the security realm.

2 Expand the roles icon.

3 Expand the role.

4 Right click one of the following:

- Allowed Users | Allow User
- Allowed Groups | Allow Group
- Allowed IDs | Allow ID

5 Supply the name of the allowed user, group, or ID.

Establishing Web service access

This section describes how to use roles to limit access to Web services and to methods/operations within a Web service.

When you add a role to a Web service or Web service operation, only the allowed users, groups, or digital IDs have access to that resource.

❖ **Adding a role to a Web service**

1 Expand the Web service collection.

2 Expand the Web service.

3 Right-click Roles and then select Add role.

4 Select a role from the list of defined roles that meets the security needs of the Web service and click OK. EAServer comes with predefined roles. For example, the “everybody” role allows unlimited access to authenticated users.

❖ **Adding a role to a Web service operation**

You can further restrict access to a Web service by assigning roles at the Web service operation/method level. For example, you could add the “everybody” role to the Web service, which allows unrestricted access to the Web service, but assign a more restrictive role to those operations that require additional restrictions.

1 Expand the Web service collection.

2 Expand the Web service.

3 Expand the Operations icon.

4 Expand the operation to which you are adding a role.

5 Right-click Roles and then select Add role.

6 Select a role from the list of defined roles that meets the security needs of the Web service operation and click OK.

If you do not assign a role to a Web service or operation, you do not need to provide a user name or password to invoke them. If you do assign a role to the Web service or the operation, you need to provide a valid user name and password for a user within the assigned role.

XML-Security

This section describes how to enable XML-Security for your Web services.

XML-Security provides a digital signature and encryption for the SOAP messages sent to and from the Web services container in EAServer. An implementation of XML Security is available at the Apache Web site at http://xml.apache.org/security.

Configuring EAServer and enabling XML-Security

EAServer must be configured with the necessary JAR files for your XML-Secure enabled Web service to work properly.

❖ **Enabling XML-Security**

1 Follow the instructions to locate and download the *xml-security-bin-1_0_4.zip* file (which contains the following JAR files) from the XML-Security package at http://xml.apache.org/security and install them in either *$JAGUAR/java/classes* (UNIX), or *%JAGUAR%\java\classes* (Windows):

- junit3.7.jar – do not install this JAR file, since one is installed as part of your EAServer installation.
- xalan.jar – verify that no other *xalan.jar* file is set in the EAServer classpath, otherwise you may experience class conflicts.
- xml-apis.jar
- log4j-1.2.5.jar – do not install this JAR file, since one is installed as part of your EAServer installation.
- xercesImpl.jar
- xmlParserAPIs.jar

- xmlsec.jar
- xmlsecSamples.jar
- xmlsecTests.jar

2 Update the EAServer classpath/bootclasspath to use the XML-Security JAR files:

- For Win2k/WinXP:

  Edit *%JAGUAR%\bin\user_setenv.bat* (create one if necessary) and add these lines:

```
set JC=..\java\classes
set EAS_CLASSPATH_P0=%JC%\junit.jar
set EAS_CLASSPATH_P0=%EAS_CLASSPATH_P0%;%JC%\log4j.jar
set EAS_CLASSPATH_P0=%EAS_CLASSPATH_P0%;%JC%\xmlsec.jar
set EAS_CLASSPATH_P0=%EAS_CLASSPATH_P0%;%JC%\xalan.jar
set EAS_CLASSPATH_P0=%EAS_CLASSPATH_P0%;%JC%\xercesImpl.jar
set EAS_CLASSPATH_P0=%EAS_CLASSPATH_P0%;%JC%\xml-apis.jar
set EAS_CLASSPATH_P0=%EAS_CLASSPATH_P0%;%JC%\xmlParseAPIs.jar
set EAS_BOOTCLASSPATH_P0=%EAS_CLASSPATH_P0%
```

- For UNIX:

  Edit *$JAGUAR/bin/user_setenv.sh* (create one if necessary), and add these lines:

```
JC=../java/classes
EAS_CLASSPATH_P0=$JC/junit.jar
EAS_CLASSPATH_P0=$EAS_CLASSPATH_P0;$JC/log4j.jar
EAS_CLASSPATH_P0=$EAS_CLASSPATH_P0;$JC/xmlsec.jar
EAS_CLASSPATH_P0=$EAS_CLASSPATH_P0;$JC/xalan.jar
EAS_CLASSPATH_P0=$EAS_CLASSPATH_P0;$JC/xercesImpl.jar
EAS_CLASSPATH_P0=$EAS_CLASSPATH_P0;$JC/xml-apis.jar
EAS_CLASSPATH_P0=$EAS_CLASSPATH_P0;$JC/xmlParseAPIs.jar
EAS_BOOTCLASSPATH_P0=$EAS_CLASSPATH_P0
```

Note By setting the EAS_CLASSPATH_PO variable, you modify the server startup script to place the XML-Security jars in the server classpath/bootclasspath first.

3 Shutdown and restart EAServer.

Exposing and deploying components as Web services

This section describes how to expose and deploy files and components as Web services:

- Deploying refers to the process of selecting a Java file or component that is located in the WST development tool (In the Package Explorer or Project view) and using one of the Deploy wizards to create the Web service and install/deploy it to a server as well as display it in the Sybase Web Services view.
- Exposing refers to the process of selecting a supported component type that already resides on the server (Sybase Web Services view) and using one of the Expose wizards to make it available as a Web service.

There are several ways to deploy and expose components as Web services depending on the options you choose, type of component, and location of component or file. For example:

- You can use the “Quickly Deploy as Web Service” or “Deploy as Web Service” wizards. Both of these wizards are available from the package explorer and from individual projects and are used to deploy a Java file as a Web service. Quickly deploying a Web service automatically uses default settings for most options.
- You can use the “Expose as Web Service” or “Quickly Expose as Web Service” wizards. Both of these wizards are available from the Other components folder of the Sybase Web Services view, and allow you to expose an existing EAServer component as a Web service.
- From the package explorer you can also select a WSDD file and choose Deploy (which is different from the wizards above).

Exposing Components as Web services

This section describes how to expose a component as a Web service.

Displaying parameter names

For the parameter names of an exposed component to display in the WST development tool:

1 Verify the parameter names are correct in the component’s IDL file.

2 Verify the stub classes are generated and compiled in debug mode before the component is exposed as a Web service.

Before generating stubs from EAServer Manager for the component you are exposing, use one of two ways to set the compiler to compile stubs using debug mode:

1 Add `-DSERVER_STUBS_DEBUG=true` to the com.sybase.jaguar.server.jvm.options property in the *server.props* file, located in the */repository/Server* subdirectory of your EAServer installation. *server* is the name of the EAServer you are modifying. For example, *jaguar.props*.

   Restart the server. This server-wide setting results in all classes being compiled with debug mode for the server.

2 Add `com.sybase.jaguar.component.javac.debug=true` to the *component.props* file for the component you are exposing, located in the */repository/component/package_name* subdirectory of your EAServer installation. *package_name* is the name of the package that contains the component you are modifying, and component is the name of the component within that package. For example, */repository/SurfSideVideoPB/n_store.props*.

   This affects the modified component only, since code generation and compilation checks the property at the component level first. If the component.javac.debug is set to "false," the classes are compiled with normal mode (non-debug) even though `SERVER_STUBS_DEBUG` is set to true in the *server.props* file.

❖ **Using the Expsose wizard to expose a Web service**

1 From the Sybase Web Services view, highlight the component that you are exposing.

2 Right-click the file and select Expose As Web Service.

3 The Expose as a Web Service wizard displays. Table 4-6 describes the Expose as a Web Service properties. Complete the information and click next to move to the next window and Finish when done.

   Error messages are logged in the server's log file and server's servlet log file. Check these files for any error conditions. For example, if you see a non-unique context path error, verify that the exposed component does not share the same Web collection name and Web service name as another exposed component, and re-expose the Web service.

Table 4-6: Exposing and Deploying Web service wizard options and properties

Window	Property	Description
General Options	Collection Name	Name of the Web service collection to which this Web service is exposed. Make sure the Web collection name and Web service name combination are unique when exposing the component as a Web service.
	Web Service Name	Name of the Web service.
	Location URL	The location where the Web service is available.
	Target Namespace	A valid Uniform Resource Identifier (URI) for the location where the WSDL document is published. The target namespace should not include the file name; WST appends the appropriate file name when the WSDL document is generated. The target namespace can be a Uniform Resource Name (URN), which is a globally unique and persistent URI. `http://www.com.sybase.webservices` is an example of a valid URI. `urn:simpleJavaClass.test` is an example of a valid URN.
	Port Type Name	Describes a collection of operation elements that define the abstract interface of the Web service. The port type name provides a unique name among all port types defined within the WSDL document. For example: `<portType name="SimplePortType">`
	Binding Name	Contains the details of how the elements of the port type name are converted to a concrete representation of the Web service by combining data formats and protocols: `<binding name="TestBinding"`
	Service Port Name	Indicates the Web service endpoint address. For example: `http://EAServer_1:8080/webservices/testPort` or `testPort`
	Implementation Class	The implementation class file to which the Web service is mapped. When you expose a component as a Web service a service implementation class file with a *.java.new* extension is created. Remove the *.new* extension and enter your business logic into the implementation file before deploying it as a Web service. Right-click the file and select Refactor | Rename to rename or remove the *.new* exention..
	Type Mapping Version	The type mapping version. Valid options are 1.1 (the default) and 1.2.
	Soap Action	The URI for the SOAPAction HTTP header for the HTTP binding of SOAP. The SOAPAction HTTP request header field can be used to indicate the intent of the SOAP HTTP request. The URI identifies the intent.

Window	Property	Description
	Binding Style	The SOAP binding style: • Document – indicates that the SOAP body contains an XML document. • RPC (remote procedure call) – indicates that the SOAP body contains an XML representation of a method/operation call. • Wrapped – a document literal variation, that wraps parameters as children of the root element.
	Soap Use	The SOAP body use: • Literal – if using a document binding style. • Encoded – if using an RPC binding style.
Method Selection	Method Name	Select the methods/operations of the Web service for which the WSDL is to be generated.
Mapping	Package Name	The package to which this Web service maps.
Deployment	Deployment Scope	The scope of the Web service deployment defined in the *server-config.wsdd*; application, session, or request (the default). For example: `<parameter name="scope" value="Application"/>` If Set to: • request – a new service implementation instance is created for any request. • application – only one shared service object is created for all requests. • session – a new object for each session-enabled client is created. If this is a Java class being deployed to EAServer there are two session classes/tables to store the objects; one for an application scoped service, and the other for a session scoped service. Once the object is created, it is saved in the appropriate table. When the next request arrives, the server checks the table for the object with the service name and reuses the object. If the object is not found, a new object for the class is created. The session scope can have a timeout setup to remove an object from the table. You can define a handler to set the timeout in the service. You can not set a timeout paramter in the *.wsdd* file, since the runtime environment does not read the timeout option in the *.wsdd* file. The application scope service timeout is not currently implemented.
Destination	Server	Highlight the server to which this Web service is deployed and exposed from.
	Web Service Collection	The Web service collection to where this Web service is contained.

Window	Property	Description
Summary		Summary of your selections. Review and click Finish to deploy, or Back to make modifications. If you are deploying a Java class, when you click Finish, the wizard creates a *.wsdd (Web service deployment descriptor)* which is an XML file that contains deployment information and location of the service implementation file, and all dependent JARs and classes. The JAR file's format matches the format that the server uses to export Web services and expects on import.

Using the quickly expose wizard

Use the quickly expose wizard to use commonly used defaults to expose a component as a Web service.

❖ **Using the quickly expose wizard to expose a Web service**

1 Highlight the package that contains the file (Java file, component, Web service, and so on) that you are deploying and exposing.

2 Right click the file and select Quickly Expose As a Web Service.

3 The Progress information window displays, indicating that the Web service is being exposed to the server to which you are connected.

Deploying Components as Web services

This section describes how to deploy a component or file as a Web service.

❖ **Using the deploy wizard to deploy a Web service**

1 From the Package Explorer or Project that contains the file to be deployed, highlight the Java file that you are deploying.

2 Right click the file and then select Deploy As Web Service.

3 The Deploy as a Web Service wizard displays. Table 4-6 describes the Deploy as a Web Service properties. Complete the information and click next to move to the next wizard and Finish when done.

Using the quickly deploy wizard

Use the quickly deploy wizard to use commonly used defaults to deploy a component as a Web service.

❖ **Using the quickly deploy wizard to deploy a Web service**

1 Highlight the component that you are deploying.

2 Right click the file and then select Quickly Deploy As a Web Service.

3 The Progress information screen displays indicating that the Web service is being deployed to the server to which you are connected. The deployed Web service also appears in the Sybase Web services view.

Generating WSDL

Web service definition language (WSDL) is the XML file that stores the metadata used to describe your Web service, defines service endpoints, and publishes information about your Web service. WSDL helps automate the generation of client proxies for Web services in a language-and platform-independent way. Like the IDL file for CORBA, a WSDL file provides the framework for client and server communication.

❖ **Generating the WSDL**

1 From a project or Package Explorer, highlight the package that contains the Java file for which you are generating WSDL.

2 Right click the file and select Generate WSDL.

3 The Generate WSDL wizard displays. Table 4-7 describes the Generate WSDL properties. Complete the information and click next to move to the next window and Finish when done.

Table 4-7: Generating WSDL wizard options and properties

Window	Property	Description
General options	Web Service Name	The Web service for which you are generating WSDL.
	Location URL	The location where the Web service is available.
	Target Namespace	A valid Uniform Resource Identifier (URI) for the location where the WSDL document is published. The target namespace should not include the file name; WST appends the appropriate file name when the WSDL document is generated. The target namespace can be a Uniform Resource Name (URN), which is a globally unique and persistent URI. `http://www.com.sybase.webservices` is an example of a valid URI. `urn:simpleJavaClass.test` is an example of a valid URN.

Window	Property	Description
	Port Type Name	Describes a collection of operation elements that define the abstract interface of the Web service. The port type name provides a unique name among all port types defined within the WSDL document. For example: `<portType name="SimplePortType">`
	Binding Name	Contains the details of how the elements of the Port type name are converted to a concrete representation of the Web service by combining data formats and protocols: `<binding name="TestBinding"`
	Service Port Name	Indicates the Web service endpoint address. For example: `http://EAServer_1:8080/webservices/testPort` or `testPort`
	Implementation Class	The name of the class file implementing the Web service.
	Type Mapping Version	The type mapping version. Valid options are 1.1 (the default) and 1.2.
	Soap Action	The URI for the SOAPAction HTTP header for the HTTP binding of SOAP. The SOAPAction HTTP request header field can be used to indicate the intent of the SOAP HTTP request. The URI identifies the intent.
	Binding Style	The SOAP binding style: • Document – indicates that the SOAP body contains an XML document. • RPC (remote procedure call) – indicates that the SOAP body contains an XML representation of a method/operation call. • Wrapped – a document literal variation, that wraps parameters as children of the root element.
	Soap uUse	The SOAP body use: • Literal – if using a document binding style. • Encoded – if using an RPC binding style.
Method Selection	Method nName	Select the methods/operations of the Web service for which the WSDL is to be generated.
Location	File Location	The location and file name (ending with *.wsdl*) of the generated WSDL file.
Summary		Summarizes your selections. Review and click Finish to generate the WSDL, or click Back to change any of your selections.

UDDI administration

From the Sybase Web Services view of the WST development tool, you can publish a WSDL document that describes your Web service and its location to a UDDI registry and unpublish from a UDDI site. See “UDDI 2.0” on page 4 for more information.

❖ **Publishing to a UDDI registry**

1 Expand the Web services folder.

2 Right-click the Web service and select Publish.

3 The Publish to UDDI wizard displays. Table 4-8 describes the Publish to UDDI properties. Complete the information and click Next to move to the next window and click Finish when you are done.

Table 4-8: Publishing to a UDDI wizard options and properties

Window	Property	Description
Select registry Profile	Registry Name	The registry to which you are connecting. From the Registry Name drop-down list, select a predefined site to which you want to log in, or select the Enter New Registry Name entry and enter a new name. You must be a registered user on the site where you log in. The registry name you select determines the default values of the query URL and the publish URL. You can modify these entries. For new names, you must provide connection information.
	Query URL	The query URL is the location from which you query the UDDI.
	Publish URL	For publishing purposes, you need both the query and publish URLs.
	User Name	The user name used for accessing the UDDI site.
	Password	The password used with the user name used to access the UDDI site.
	Save Profile	Use this button to save a profile. It will be added to the Registry Name drop-down list for easy access.
	Delete Profile	Use this button to delete a profile that you no longer require.
	Ping	Use this button to test your profile connection. You should be able to ping before moving on to the other windows.
Business Information	Name	The name of the organization name by which this UDDI entry is known.
	Description	A description of the organization.
	Use Existing tModel Key	Your business model. The tModel is an abstract description of a particular specification or behavior to which the Web service adheres.
	Service Description	A description of the service the business provides.

Window	Property	Description
	Retrieve Existing Businesses and tModels from Registry	You can use this button to query the UDDI registry for tModel and business information instead of entering this information manually.
Summary		Displays a summary of your selections. Click Finish to publish to the UDDI site, or click Back to change your selections.

❖ **Unpublishing from a UDDI**

1 Expand the Web services folder.

2 Right-click the Web service and select Unpublish.

3 The Unpublish from UDDI wizard displays. Table 4-9 describes the Unpublish to UDDI properties. Complete the information and click Next to move to the next window and Finish when done.

Table 4-9: Unpublishing from a UDDI wizard options and properties

Window	Property	Description
Select Publishing Profile	Registry Name	The registry to which you are connecting. From the Registry Name drop-down list, select a predefined site to which you want to log in, or select the Enter New Registry Name entry and enter a new name. You must be a registered user on the site where you log in. The registry name you select determines the default values of the Query URL and the Publish URL. You can modify these entries. For new names, you must provide connection information.
	Query URL	The query URL is the location from which you query the UDDI.
	Publish URL	For publishing and unpublishing purposes, you need both the query and publish URLs.
	User Name	The user name used for accessing the UDDI site.
	Password	The password used in connection with the user name used to access the UDDI site.
	Save Profile	Use this button to save a profile. It will be added to the Registry Name drop-down list for easy access.
	Delete Profile	Use this button to delete a profile that you no longer require.
	Ping	Use this button to test your profile connection. You should be able to ping before moving on to the other windows.
Select UUIDs	Name of UUID	A list of universally unique identifier (UUID) that identifies the UDDI entry for all of your UDDI entries is displayed. Check only those entries that you want to unpublish.
	Check for Empty Businesses	Select to check for empty businesses. An empty business may not have a UDDI associated with it.

Window	Property	Description
	Check for Unused tModels	select to check for unused tModels. An unused tModel may not have a UDDI associated with it.
Selected UUID Details	Service Details	The service details of your UDDI entry as identified by the UUID identifier.
Summary		Displays a summary of your selections. Click Finish to unpublish from the UDDI site, or click Back to change your selections.

Other components

The Other Components folder shows components located on the server to which you are connected that can be converted to the SOAP message format. In other words the Other Components folder contains components capable of being exposed as Web services.

There may be components on the server to which you are connected that, in order to make available, you must modify the component. For example, a component can be exposed as a Web service only if it is stateless. You can use EAServer Manager to mark the component stateless. Until then, it does not display in the Other Components folder.

See “Exposing Components as Web services” on page 48 and “Using the quickly expose wizard” on page 52 for information about deploying other components as Web services.

CHAPTER 5

Web Console—Web Services

The Sybase Web console is a Web based management console that provides plug-in support, for example Web Services Toolkit. This chapter describes how to use the Web console to manage Web services. For information about using the Web console to manage the private UDDI server, and publish to UDDI registries, see Chapter 6, "Web Console—Registry Services."

Introduction

The Web console is a J2EE Web application, distributed with EAServer, which allows you to view and manage Web services contained in EAServer.

You can perform many of the same functions described in this chapter using wstool and wstkeytool commands or the WST development tool interface.

❖ **Connecting to the Web console**

To connect to the Web console, EAServer must be running.

1 In your Web browser, go to the Web console application located at at http://hostname:8080/WebConsole, where *hostname* is where EAServer runs.

2 Enter a user name and password and click Login to connect to the Web console. For complete access use `jagadmin` as the user name. The default password for `jagadmin` is blank.

Logging in to the Web console versus logging into Sybase Central

There are some differences you should be aware of regarding logging in to the Web console versus EAServer using Sybase Central:

- Sybase Central does not require you to log in. Sybase Central is not hosted in an application server, has no security in and of itself, and stores its information based on operating system authentication information, not a separate authentication system. Sybase Central stores its information related to who the user logged in to the operating system as and what machine you logged in from.
- Web console stores its information based on the user logged in to the Web console. This allows you to retain your preferences no matter what machine you logged in from or who you logged in to the operating system as.

If the Web console is hosted in the same server that you want to connect to, you must log in to both the Web console and the server, using, in most cases, the same user name and password. You can, however, mark your server profile as "auto connect" (described in Table 5-1 on page 65), which means you only need to log in to the Web console, and it automatically logs in to the server.

Web console security and access

The Web console uses Web application security. A user must be created and added to the Admin role to have access to all of the Web console's features. By default, jagadmin is a member of the Admin role. There is also a role named WebConsole_ReadOnly. Users that belong to this role have read only access to the Web console; changes made to user's preferences or any profiles are not saved.

Adding an authentication service to EAServer

By default, EAServer does not have a default authentication service installed. Only the jagadmin user is authenticated. Authentication is enabled either by installing a custom authentication service (see *html/ir/CtsSecurity__AuthService.html* in your EAServer installation directory), or by installing a JAAS login module.

EAServer provides a sample server side JAAS login module. To install the sample JAAS login module in EAServer:

1 From EAServer Manager, select Server Properties | Security, and set the JAAS configuration file to *$JAGUAR/html/classes/Sample/JAAS/jaas.cfg*.

2 Select Server | Advanced | All Properties tab, and if necessary, define a new property com.sybase.jaguar.server.jaas.section. Set the value of this property to UsernamePasswordBased.

3 Restart the server.

The *jaas.cfg* file points to your $*JAGUAR/html/classes/Sample/JAAS/users.xml* file, which contains two users, jagadmin and anonymous. By default, jagadmin is a member of the Admin role and anonymous is a member of the WebConsole_ReadOnly role. If you log in to the Web console as anonymous, and use the password "pwd," you see that changes made to server profiles, preferences, and so on, persist only for the length of your session (until you log out or the session times out).

Note Check with your EAServer administrator before making any changes. For more information see the JAAS Web site at http://java.sun.com/products/jaas/.

---

users.xml file

This modified *users.xml* file illustrates how you can implement authentication using a JAAS login module:

```
<?xml version="1.0" encoding="UTF-8" ?>
<user_info policy="MD5">
<user>
 <name>jagadmin</name>
  <description>com.sybase.jaguar.server.jaas.section=UsernamePasswordBased
 </description>
</user>
<user>
 <name>anonymous</name>
```

```
  <description />
  <password>pwd</password>
</user>
<user>
  <name>new_user</name>
  <description />
  <password>pwd</password>
</user>
</user_info>
```

Disabling JavaScript

You can run the Web console with JavaScript disabled, although the context-sensitive menus require JavaScript. After enabling JavaScript in the browser, log in to the console again.

The wizard functionality of the Web console assumes that JavaScript is enabled. If disabled, there are minor differences in wizard behavior. These differences include:

- The context menus do not appear. You must instead use the menu below the tree view.
- Combination dialogs do not work.
- Wizards and property sheets do not indicate default values.
- Wizards are always shown in the right pane, even if the preference is set for pop-up windows.

Browser support

The console runs best with Internet Explorer version 5.5 and higher, Netscape version 7.*x* and higher, and Mozilla version 1.0 and higher.

Authentication time-out in EAServer

The default EAServer authentication time-out is set to 1 hour or 3600 seconds. If the console remains unused for longer than this, you must log in again. You can adjust the time-out property by modifying the com.sybase.jaguar.server.authtimeout property from EAServer Manager.

Session time-out in EAServer

The default session time-out for the console is to never time-out. The associated properties are:

- com.sybase.jaguar.webapplication.session-config.session-timeout
- com.sybase.jaguar.webapplication.session-config

Plug-in, domain, display, and server administration

This section describes how to use the Web console to manage the Sybase Web services plug-in, domains, and servers to which Web service collections belong. It also describes how to modify preferences which determines how Web console wizards, nodes, and the interface are displayed.

❖ **Modifying preferences**

1 Highlight the Preferences icon. The General properties tab displays.

- Show wizards in – determines where and how the Web console display wizards.
- Show category nodes in – determines how category nodes display.
- Look and feel – determines the look and feel of the interface.

2 Click Apply when done.

❖ **Defining Web Services Toolkit plug-in parameters**

You can establish default values for Web Services Toolkit, which allows you to manage the connection information for server profiles.

1 Click the Plug-ins folder.

2 Highlight the Sybase Web Services Toolkit folder.

3 Complete the General properties section to establish server profile values. Table 5-1 on page 65 describes the properties.

❖ **Creating a domain**

1 Right-click the Web Services Toolkit icon and select Create Domain.

2 The Create Domain wizard appears. Enter the information as instructed by the wizard and click Next. When finished, click Finish. The new domain appears.

❖ **Deleting a domain**

- Right-click the domain to delete and select Delete.

❖ **Creating a server profile**

1 Right-click the domain in which the server profile you are creating belongs and select Create Server Profile.

2 The Create Server Profile wizard appears. Enter the information as instructed by the wizard and click Next. When finished, click Finish. The new server profile appears in the domain in which it was created. Table 5-1 on page 65 describes the server profile properties.

❖ **Connecting to a server**

You can connect only to those servers for which you have a server profile.

1 Expand the domain in which the server profile you are connecting belongs.

2 Right-Click the server profile you want to connect to and choose Connect from the menu.

3 If the connection fails, click the Connection Details tab to review the connection details. Table 5-1 on page 65 describes the connection properties.

❖ **Restarting, stopping, deleting, or disconnecting from a server profile**

- Right-click the server and click the action you want to perform:
  - Restart – restarts the server to which you are connected.
  - Stop – stops the server to which you are connected.
  - Delete – deletes the server profile for the server to which you are connected.
  - Disconnect – disconnects from the server to which you are connected.

Table 5-1 describes plug-in, domain, and server properties.

Table 5-1: Plug-in, domain, and server profile properties

Property	Description
Select Domain (plug-in property only)	The domain for the plug-in.
Select Server (plug-in property only)	The server for the plug-in.
Server Profile Name (server property only)	The name of the server profile that you are creating.
Machine Name	The name of the host machine where the server resides.
Protocol	The protocol used to connect to the server; “http” or “https.”
HTTP Port	The port number of the host used to connect to the server; for example, `8080`.
User ID	The user name used to connect to the server. `jagadmin` is the default. Use `jagadmin` or another member of the Admin role to connect to the Web services container for access to all of Web Console’s functions. Members of the WebConsole_ReadOnly role have limited access to the Web Console.
Password	The password of the user connecting to the server. The default for `jagadmin` is blank.
Auto Connect on Console Login	Select this box to connect to this profile automatically when you log in to the Web console.

A node can be a plug-in, domain, server profile, Web service collection, Web service, and so on. If node information changes, or you want to reset the view, right-click the node you are refreshing and select Refresh.

Web service collection administration

You can create and maintain Web service collections on each server being administered by the Web console.

❖ **Viewing or modifying Web service collection properties**

1 Expand the server that contains the Web service collection whose properties you are viewing or changing.

2 Highlight the Web service collection. The Web console displays General and Web Service tabs. Table 5-2 on page 67 describes the Web service collection properties.

3 Make any changes and click Accept when done or Reset to ignore your changes.

❖ **Exporting a Web service collection**

You can export a Web service collection to a WAR file. Once exported, you can import and deploy the WAR file to and from other Web service servers.

1 Expand the server that contains the Web service collection you are exporting.

2 Right-click the Web service collection and select Export.

3 Follow the wizard instructions to export the Web service collection. By default, the Web service collection is exported to a file named *wscoll.war*, where *wscoll* is the name of the Web service collection.

❖ **Importing a Web service collection**

You can import a Web service collection from a WAR file into the Web services server to which you are connected.

1 Expand the server to which you want to import the Web service collection.

2 Right-click the Web Service Collection folder and select Import.

3 Follow the wizard instructions to import the Web service collection. Use Browse to locate the WAR file that contains the Web service collection. “ws” is the default Web service collection, if not specified.

4 When you click Finish, the Web service collection is imported and displays under the Web Service Collection folder.

❖ **Deleting a Web services collection**

To delete a Web collection and all of the Web services it contains:

1 Expand the server that contains the Web service collection you are deleting.

2 Right-click the Web service collection and select Delete.

Table 5-2 describes the Web services collection properties.

Table 5-2: Web service collection properties

Property	Description
Name	The name of the Web services collection.
Description	A description of the Web services collection.
Realm	The realm (if any) to which the Web collection belongs. A realm defines the scope of authentication and authorization, and is also referred to as a security realm.
HTTP Authentication Method	The authentication method (if any) used by your Web service collection. Authentication method choices are the same as used by Web applications. See Chapter 3, "Web Application Security" in the *EAServer Security Administration and Programming Guide* for more information.

Web service administration

This section describes the procedures used to manage individual Web services.

❖ **Viewing or modifying Web service properties**

1 Expand the Web service collection that contains the Web service you want to view or modify.

2 Highlight the Web service.

3 Select the General tab to view the Web service properties. See Table 5-3 on page 68 for a description of the Web service properties.

4 Select the WSDL tab to view the WSDL for this Web service.

❖ **Activating a Web service**

If a Web service is deactivated, the Web service icon has an "X" through it. You must activate the Web service to make it available to clients.

1 Expand the Web service collection that contains the Web service you want to activate.

2 Right-click the Web service and select Activate.

❖ **Deactivating a Web service**

If a Web service is activated, the Web service icon appears without an "X" through it. Deactivate a Web service to make it unavailable to clients.

1 Expand the Web service collection that contains the Web service you want to deactivate.

2 Right-click the Web service and select Deactivate.

❖ **Deleting a Web service**

This procedure deletes a Web service from a Web service collection.

1 Expand the Web service collection you are deleting.

2 Right-click the Web service and select Delete.

Table 5-3 describes the Web service properties.

Table 5-3: Web service properties

Property type	Property	Description
General	Name	The name of the Web service.
	Description	A description of the Web service.
	Implementation type	The type of component, class, or file that implements the Web service.
	Implementation class name	The name of the class file implementing the Web service.
	Style	The SOAP binding style: • Document – indicates that the SOAP body contains an XML document. • RPC (remote procedure call) – indicates that the SOAP body contains an XML representation of a method call.
	Use	The SOAP body use: • Literal – if using a document binding style. • Encoded – if using an RPC binding style.
	Service URL	The path, URL, or endpoint from which the Web service can be accessed.

Web service operation management

This section includes the procedures used to manage the operations (methods) of a Web service.

Overloaded methods

If you deploy a Web service that contains overloaded methods, the Web console displays only the first method of the overloaded method. Allowing or disallowing access to the method, affects all overloaded methods.

For example, if the Web service contains an overloaded method that contains the methods echo(String, String) and echo (String), the GUI displays only echo (String, String) twice, but the allowed/disallowed operation affects both echo(String, String) and echo(String).

❖ **Viewing or modifying Web service operation properties**

1. Select the Web service collection and Web service you want to view or modify.
2. Highlight the Operations folder.
3. Select the General tab to view the Web service Operations properties. See Table 5-4 on page 70 for a description of the Web service properties.

❖ **Allowing an operation**

1. Select the Web service collection and Web service that contains the operation to which you want to allow client access.
2. Highlight the Operations folder.
3. Right-click the operation and select Allow.

❖ **Disallowing an operation**

1. Select the Web service collection and Web service that contains the operation to which you want to disallow client access.
2. Highlight the Operations folder.
3. Right-click the operation and select Disallow.

❖ **Invoking an operation**

1. Select the Web service collection and Web service that contains the operation you want to invoke.
2. Highlight the Operations folder.
3. Right-click the operation and select Invoke.
4. If a role is assigned to the operation, you may need to provide a user name and password to invoke the operation:

   If a role is not assigned to a Web service operation, you do not need to provide a user name or password to invoke it. If a role is assigned to the Web service operation, you must provide a valid user name and password for a user within the assigned role.

Table 5-4 describes the Web service operation properties.

Table 5-4: Web service operation properties

Property type	Property	Description
General	Name	The name of the operation.
	Description	A description of the Web service operation.
	Binding Style	Specify the SOAP binding style: • Document – indicates that the SOAP body contains an XML document. • RPC (remote procedure call) – indicates that the SOAP body contains an XML representation of a method call.
	Return Type	Specifies the return type of the method.
	Is Return Value In Response Message	True or false.
	SOAP Action	The URI for the SOAPAction HTTP header for the HTTP binding of SOAP. The SOAPAction HTTP request header field can be used to indicate the intent of the SOAP HTTP request. The URI identifies the intent.
	Message Operation Style	Document, RPC, or wrapped.
	Is Allowed	True or false. Determines whether or not the method is available to a client as a Web service endpoint.

Web service parameter management

This section describes the procedures used to manage the parameters for a given method or operation of a Web service.

❖ **Viewing or modifying parameters**

1 Select the Web service collection and Web service you want to view or modify.

2 Highlight the Operations folder.

3 Highlight the operation of interest.

4 Click the Parameters folder.

5 Highlight the parameter of interest.

6 Select the General tab to view the parameter properties. See Table 5-5 for a description of the parameter properties.

Table 5-5: Web service parameter properties

Property	Description
Name	Name of the parameter.
Type	The type of parameter. The type cannot be edited.
Mode	The mode of the parameter, “in”, “out”, or “inout”.
Order	The order of the parameters. If there is only one parameter, the order is “0”.

UDDI administration

This section describes how to publish information about your Web service and its location to a UDDI registry and unpublish from a UDDI site.

❖ **Publishing to a UDDI registry**

1 Expand the Web Service Collection folder.

2 Expand the Web service collection to which the Web service you are publishing belongs.

3 Right-click the Web service and then select Publish to UDDI.

4 The Publish to UDDI wizard displays. Table 5-6 describes the Publish to UDDI properties. Complete the information and click Next to move to the next window. Click Finish when done.

Table 5-6: Publishing to a UDDI wizard options and properties

Window	Property	Description
Publish to UDDI	Registry Profile	The registry to which you are connecting. From the Registry Profile drop-down list, select a predefined site to which you want to log in or select the Enter New Registry Profile entry and enter a new name. You must be a registered user on the site where you log in. The registry profile you select determines the default values of the registry name, query URL, and the publish URL. You can modify these entries. For new profiles, you must provide connection information.
	Registry Name	The name of the registry to which you are connecting.
	Query URL	The location from which you query the UDDI registry.
	Publish URL	For publishing purposes, you need both the query and publish URLs.
	User Name	The user name for accessing the UDDI site.
	Password	The password used with the user name used to access the UDDI site.

Window	Property	Description
	Save Profile	Save a profile. It will be added to the Registry Name drop-down list for easy access.
	Delete Profile	Delete a profile that you no longer require.
	Ping	Test your profile connection. You should be able to ping before moving on to the other wizards.
Business Information	Name	The name of the organization name by which this UDDI entry is known.
	Description	A description of the organization.
	Use Existing tModel Key	Your business model. The tModel is an abstract description of a particular specification or behavior to which the Web service adheres.
	Retrieve Existing Businesses and tModels from Registry	Query the UDDI registry for tModel and business information instead of entering this information manually.
	New Business	Add a new business name and information for this Web service.
Summary		Displays a summary of your selections. Click Finish to publish to the UDDI site, or click Back to change your selections.

❖ **Unpublishing from a UDDI**

1 Expand the Web Service Collections folder.

2 Expand the Web service collection that contains the Web service you are unpublishing.

3 Right-click the Web service and then select Unpublish from UDDI.

4 The Unpublish from UDDI wizard displays. Table 5-7 describes the properties. Complete the information and click Next to move to the next window. Click Finish when done.

Table 5-7: Unpublishing from a UDDI wizard options and properties

Window	Property	Description
Unpublish from UDDI	Registry Profile	The registry to which you are connecting. From the Registry Profile drop-down list, select a predefined site to which you want to log in, or select the Enter New Registry Profile entry and enter a new name. You must be a registered user on the site where you log in. The registry profile you select determines the default values of the registry name, query URL, and publish URL. You can modify these entries. For new profiles, you must provide connection information.
	Registry Name	The name of the registry to which you are connecting.
	Query URL	The location from which you query the UDDI registry.

Window	Property	Description
	Publish URL	For publishing and unpublishing purposes, you need both the query and publish URLs.
	User Name	The user name for accessing the UDDI site.
	Password	The password used in connection with the user name used to access the UDDI site.
	Save Profile	Save a profile. It is added to the Registry Name drop-down list for easy access.
	Delete Profile	Delete a profile that you no longer require.
	Ping	Test your profile connection. You should be able to ping before moving on to the other wizards.
Select Published UUIDs to be Unpublished	Name of UUID	Click the Get Published Services Named *WebServiceName* (where *WebServiceName* is the name of the Web service you are unpublishing). This returns a list of universally unique identifier (UUID) that identifies the UDDI entry for this Web service. Select only those entries that you want to unpublish.
	Unpublish the Business	Click this checkbox to unpublish business information for this Web service.
	Unpublish the tModel	Unpublish tModel information for this Web service.
Summary		Displays a summary of your selections. Click Finish to unpublish from the UDDI site, or click Back to change your selections.

Type mappings

Each Web service contains a Type Mapping folder that contains the type mappings used to transfer data between service endpoints.

For complete information, see Chapter 3, “Components, Datatypes, and Type Mappings.”

Handlers

See “Handlers” on page 42 for information about handlers.

Managing security realms and roles

EAServer contains a default security realm. The security realm is a container used to store the roles used to allow and limit access to your Web services. When you connect to EAServer from the Web console, you see the security realm.

❖ **Refreshing a security realm**

If you add a role to a security realm or make any other changes outside the current session of the Web console, you must refresh the realm to see those changes.

- Right-click the security realm and select Refresh.

Roles

EAServer's authorization model is based on roles. Roles control access to that Web service, and are located in the Roles folder under each Web service.

You can also define roles at the Web service operation level. Establishing roles at the operation/method level gives you even greater access control of Web services resources.

You can define new roles from the Web console. See the *EAServer Security Administration and Programming Guide* for information about creating and installing roles in EAServer.

❖ **Adding a role to a Web service**

1. Select the Web service collection and Web service to which you are adding a role.
2. Right-click the Roles folder and select Assign Roles.
3. Click Next and follow the instructions to add a role to the Web service. The Role wizard displays a list of existing roles contained on the server to which you are connected in the Available roles box. Highlight a role and click the arrow button to place it in the assigned roles box.

You can also enter a name of a newly created role and click Add Newly Available Role to add this role to the Available roles box, which can then be assigned to the Web service.

4 Click Next to review the roles that you have assigned to the Web service. Click Finish, Back (to change your settings), or Cancel.

❖ **Removing a role from a Web service**

This procedure allows you to remove a role that is assigned to a Web service. It does not delete the role from the server.

1 Select the Web service collection and Web service from which you are removing a role.

2 Right-click the Roles folder, select the role you want to remove, and select Remove Role.

3 Click Next and follow the wizard instructions to remove the role from the Web service. Click Finish, Back (to change your settings), or Cancel.

❖ **Adding a role to a Web service operation**

1 Select the Web service collection and Web service to which the operation belongs.

2 Select the Operations folder and the operation to which you are adding a role.

3 Right-click the Roles folder and select Assign Roles.

4 Click Next and follow the instructions to add a role to the Web service operation. The Role wizard displays a list of existing roles in the Available roles box. Highlight a role and click the arrow button to place it in the Assigned Roles box.

You can also enter a name of a newly created role and click Add Newly Available Role to add this role to the Available roles, which can then be assigned to the Web service operation.

5 Click Next to review the roles that you have assigned. Click Finish, Back (to change your settings), or Cancel.

❖ **Removing a role from a Web service operation**

This procedure allows you to remove a role that is assigned to a Web service operation. It does not delete the role from the server.

1 Select the Web service collection and Web service to which the operation belongs.

2 Select the Operations folder and the operation from which you are removing a role.

3 Right-click the Roles folder, select the role you want to remove, and select Remove Role. The Remove Role wizard displays.

4 Click Next and follow the wizard instructions to remove the role from the Web service operation. Click Finish, Back (to change your settings), or Cancel.

Runtime monitoring

You can monitor statistics and performance of your Web services from within the Runtime Monitoring folder.

❖ **Monitoring a Web service**

1 Select the server profile, then select RunTime Monitoring folder | Web Services folder | Web service collection. Select the Web service that you want to monitor.

2 Right-click the Web service, then select Enable Service Statistics.

3 The Web console displays the available monitoring options. Select:

- Data Throughput – allows you to view various throughput statistics for this Web service, including:
  - Hits – number of times the Web service has been accessed.
  - Input statistics – the total number of bytes, message size, and message timestamp of an incoming message directed to the Web service.
  - Output statistics – the total number of bytes, message size, and message timestamp of outgoing messages generated by the Web service.
  - Number of failed requests – the total number of failed Web service requests.
- Performance – allows you to view various performance statistics for this Web service, including:

- Hits – number of times the Web service has been accessed.
- Enabled time and total time – when and the amount of time since monitoring was enabled.
- Dispatch measurements – various statistics that monitor the performance by calculating various dispatch times, that is, how quickly the server is responding to a client's Web service request.

You should also view the EAServer log files regarding Web service messages and errors.

Non-Web service components

The Non-Web Service Components folder contains components that are hosted on the server to which the Web console is connected and capable of being exposed as Web services.

❖ **Exposing a non-Web service component**

1 Expand the Non-Web Service Component folder.

2 Expand the package that contains the component you want to expose as a Web service.

3 Right-click the component and select Expose as Web Service. Follow the instructions to expose the component as a Web service. Table 4-6 on page 50 describes the properties. When you click Finish, the Web service is exposed in the Web service collection you entered.

CHAPTER 6

Web Console—Registry Services

This chapter describes how to use the Sybase Management Web console to administer information contained in the private UDDI server, and publish to a UDDI registry.

For information about using the Web console to manage Web services, see Chapter 5, “Web Console—Web Services.”

For information about starting, and configuring the private UDDI server, see Chapter 7, “The Private UDDI Server.”

Introduction

This portion of the Web console consists of two independent parts:

- An administration console for the private UDDI server – Sybase provides a private UDDI registry as part of Web services. The private UDDI registry is an internal registry that provides an index of Web services in a particular domain, behind the firewall and isolated from the public network. This ensures that access to both the administrative features and registry data are secured. Data in the registry is not shared with any other registries.
- A browser capable of searching and publishing to any UDDI registry.

Using the Web console

This section describes how to connect to and navigate the Web services registry section of the Web console.

❖ **Connecting to the Web console**

To connect to the Web console, EAServer must be running.

1 In your Web browser go to the Web console application located at at http://hostname:8080/WebConsole, where *hostname* is the name of the host on which EAServer runs.

2 Enter a user name and password and click Login to connect to the Web console. Use `jagadmin` as the user name. The default password for jagadmin is blank.

Navigating the console and managing resources

Navigate the Web console by selecting the desired option or folder in the left pane. UDDI administration functions and property sheets are located in the UDDI Registries folder within Web Services Registries, and include:

- UDDI on localhost – this is the private UDDI registry.
- Registry Administration – includes defining and managing registries. See "UDDI registry profile administration" on page 81.
- Search – search UDDI registries. See "Inquiries and searches" on page 82.
- Publish – publish business information to UDDI registries. See "Publishing" on page 84.

Note For all property sheets, the contents cannot be edited if they are properties of a node rooted in the Search hierarchy. If they are properties of a node rooted in the Publish hierarchy, they can be edited, unless they are keys, which can never be edited. Tables that can be edited include a Delete check box column and an Add button. Property sheet pages that can be edited display Apply and Cancel buttons at the bottom of the page.

UDDI administration

This section describes how to administer UDDI registries including, the private UDDI server from the Web console.

Registry profile information (URLs, user IDs, passwords, and so on) and the users allowed to access them are stored in a repository accessible by the Web console, along with the information necessary to publish a Web service to a registry.

Note You must install JDK 1.4 to run the UDDI server. A typical EAServer installation includes JDK 1.4 and installs the UDDI server.

UDDI registry profile administration

You can create, modify, or delete UDDI registry profiles for the private UDDI server on the machine to which you are connected, where you can publish business and service information.

❖ **Creating a UDDI registry profile**

1 Right-click the Web Services Registry icon and select Create Registry Profile.

2 Follow the wizard instructions to create the UDDI registry profile. See Table 6-1 on page 82.

❖ **Connecting to a UDDI registry profile**

1 Expand the Web Services Registry icon.

2 Right-click the registry profile to which you want to connect and select Connect. The Web console attempts to connect to the registry with the information provided when it was created. See Table 6-1 on page 82. If the Web console successfully connects to the registry, the Search and Publish folders display. If you want the profile to connect to the private UDDI registry server when you connect to the profile, click “Automatically connect to registry Server” checkbox available from the Connection Details window.

❖ **Deleting or modifying a UDDI registry profile**

1 Expand the Web Services Registry icon.

2 Right-click the registry profile you want to delete and select Delete.

Table 6-1: UDDI registry profile properties

Wizard	Property	Description
Create UDDI registry profile	Registry name	The name of the registry you are creating or modifying.
	Query URL	The location from which you query the UDDI. The default query URL for the private UDDI registry is at http://localhost:8080/uddi-server/inquiry.
	Publish URL	To publish, you need both query and publish URLs. The default publish URL for the private UDDI registry is at http://localhost:8080/uddi-server/publish
	User name	The user name used for accessing the UDDI site.
	Password	The password used in connection with the user name used to access the UDDI site.
Summary		Displays a summary of your selections. Click Finish to create the UDDI site, or click Back to change selections.

Searching and publishing to UDDI registries

This section describes how to search, query, and publish to UDDI registries.

Inquiries and searches

From the Web console, you can query the private UDDI registry as well as external UDDI registries to locate potential clients or suppliers based on business type, categories, services, and so on. Locate information about specifications, protocols, and namespaces of services and classification systems through the tModels that describe and identify them.

Searching UDDI registries

This section describes how to search a registry by business, service, or tModel entry.

❖ **Searching a registry**

1 Select the Search folder.

2 Complete the search options for the type of search you want to perform and click Search. Table 6-2 describes the search properties.

3 When the search completes, click the Results folder to view the results.

4 Click any of the items returned from the search to view additional information about a business, service, or tModel.

Table 6-2: Search properties

Search type	Options	Description
Business	Business name	Enter a name of a business for which you are searching.
	Sort by name	Select this check box and click Ascending or Descending, depending on the order you want to display the businesses.
	Sort by date	Select this check box to sort businesses by the date they were created or modified.
	Case sensitive	Considers case when performing a search.
	Exact match	Search only for those businesses that exactly match the Business name.
	Advanced options	Advanced search options allow you to search by: Categories – can be used in searches to locate information in a registry based on business, service, or tModel category. Identifiers – an industry-standard identifier is unique to a business or tModel.
	Add Category	Add a category to this business. See “Categories” on page 88 for more information about categories.
	Add Identifier	Add an identifier to this business. See “Identifiers” on page 89 for more information about identifiers.
Service	Service name	Enter a name of a service for which you are searching.
	Sort by name	Select this check box and click Ascending or Descending depending on the order you want to display the service.
	Sort by date	Select this check box to sort services by the date they were created or modified.
	Case sensitive	Consider case when performing a search.
	Exact match	Search only for those services that exactly match the Business name.
	Add Category	Add a category to this business. See “Categories” on page 88 for more information about categories.
tModel	tModel name	Enter a name of a tModel for which you are searching.

Search type	Options	Description
	Sort by name	Select this check box and click Ascending or Descending, depending on the order you want to display the tModel.
	Sort by date	Select this check box to sort tModels by the date they were created or modified.
	Case sensitive	Consider case when performing a search.
	Exact match	Search only for those services that exactly match the tModel name.
	Add Category	Add a category to this business. See “Categories” on page 88 for more information about categories.
	Add Identifier	Add an identifier to this business. See “Identifiers” on page 89 for more information about identifiers.

Publishing

You can publish and manage information about your business, its organization, Web services, or other services offered from the Web console to a UDDI registry. After the business or service is published, the information is accessible to the clients of the registry.

Businesses

A UDDI registry allows you to describe your business and publish information about the services of that business. You can list categories and identifiers to which the business belongs, which provides additional ways for clients to search your business for particular services. You can supply contact information so that your business can be located easily.

❖ Adding a business

1 Expand the Publish folder.

2 Right-click the Published Businesses folder and select Add Business.

3 Follow the Add Business Entity wizard to add a business. See Table 6-3 on page 85 for a description of the business properties.

❖ Deleting a business

1 Expand the Publish folder.

2 Expand the Published Businesses folder.

3 Right-click the business you want to delete and select Delete.

Table 6-3: Business properties

Tab	Property	Description
Business	Name	The name of the published business.
	Description	A description of the business.
	Key	A unique key that is generated when the business is registered.
	Related businesses	The key of any related or similar businesses.
Summary		Displays a summary of your selections. Click Finish to create the business, or click Back to change selections.

For each published business, you can add a:

- Service – see “Services” on page 85
- Contact – see “Contacts” on page 90
- Discovery URLs – see “Discovery URLs” on page 91
- Categories – see “Categories” on page 88
- Identifiers – see “Identifiers” on page 89

Services

Web services reside in businesses. Web services can be organized into categories using identifiers, and can include access information that provides easy access to clients.

❖ Adding a service

1 Expand the Publish folder.

2 Right-click the Published Services folder and select Add Service. Or, to add a service to an existing business, expand the Published Businesses folder and select the Published Services folder within it and select Add Service.

3 Follow the Add Service Entity wizard to add a service. See Table 6-4 on page 86 for a description of the service properties.

❖ Deleting a service

1 Expand the Publish folder.

2 Expand the Published Services folder.

3 Right-click the service you want to delete and select Delete.

Table 6-4 describes the service properties.

Table 6-4: Service properties

Tab	Property	Description
General	Name	The name of the service.
	Description	The description of the service.
	Language	The language name and description.
Summary		Displays a summary of your selections. Click Finish to add a service, or click Back to change selections.

For each published service, you can add:

- Bindings – see “Bindings” on page 87
- Categories – see “Categories” on page 88
- Identifiers – see “Identifiers” on page 89

tModels

tModels reference a technical specification or description of a Web service. They provide descriptions of Web services that define service types. Each tModel includes a unique identifier (key) and points to a specification that describes the Web service. tModels provide a common point of reference that allows you to locate compatible services.

❖ Adding a tModel

1 Expand the Publish folder.

2 Right-click the Published tModels folder and select Add tModel.

3 Follow the Add tModel Entity wizard to add a tModel. See Table 6-5 on page 87 for a description of the tModel properties.

❖ Deleting a tModel

1 Expand the Publish folder.

2 Expand the Published tModels folder.

3 Right-click the tModel you want to delete and select Delete.

Table 6-5: tModel properties

Tab	Property	Description
General	Name	Name of the tModel.
	Description	Description of the tModel.
	Language	The language name and description.
Summary		Displays a summary of your selections. Click Finish to create the tModel, or click Back to change selections.

For each published tModel, you can add a:

- Discovery URL– see "Discovery URLs" on page 91
- Categories – see "Categories" on page 88
- Identifiers – see "Identifiers" on page 89

Additional registry information for published businesses, tModels, and services

After you have published businesses, tModels, or services to a registry, you can add additional information to each.

Bindings

Bindings are the mechanisms that bind the abstract definition (overview document, or description) of a Web service to the concrete representation (access point) of that service.

❖ **Adding a binding to a service**

1 Expand the Publish folder.

2 Expand the Published Services folder.

3 Expand the service to which you are adding a binding.

4 Right-click the Bindings folder and select Add ServiceBinding.

5 Follow the Add ServiceBinding Entity wizard to add a binding. See Table 6-6 on page 88 for a description of the binding properties.

❖ **Deleting a binding from a service**

1 Expand the Publish folder.

2 Expand the Published Services folder.

3 Expand the service to which the binding you are deleting belongs.

4 Expand the Bindings folder.

5 Right-click the binding you want to delete and select Delete.

Table 6-6 describes the binding properties.

Table 6-6: Binding properties

Tab	Property	Description
General	Description	A description of the binding.
	Access point	An address for accessing a Web service must be a valid URL.
	Language	The language name and description.
Summary		Displays a summary of your selections. Click Finish to create the binding, or click Back to change selections.

Categories

Each business classification system has codes for the various categories. A categories scheme allows you to group registry entries by a given category. For example, large businesses that conduct a variety of business may be sorted by several classifications. A company might sell computer hardware and software. Such a business might be listed with several classifications, such as computer training, data processing services, and software publishers, and so on. Each business classification also has a corresponding key.

❖ **Adding a category to a service, tModel, or business**

1 Expand the Publish folder.

2 Expand the Published Services, tModel, or businesses folder.

3 Expand the service, tModel, or business for which you are adding a category.

4 Right-click the Categories folder and select Add Category.

5 Follow the Add Category Entity wizard to add a category. See Table 6-7 on page 89 for a description of the category properties.

❖ **Deleting a category from a service, tModel, or business**

1 Expand the Publish folder.

2 Expand the Published Services, tModels, or Businesses folder.

3 Expand the service, tModel, or business to which the category you are deleting belongs.

4 Expand the Categories folder.

5 Right-click the category you want to delete and select Delete.

Table 6-9 describes the category properties.

Table 6-7: Category properties

Tab	Property	Description
General	Categorization Scheme	Select the categorization scheme to use with the Web service, tModel, or business.
	Name	The name of the category.
	Value	Each category has a corresponding value.
	Key	A unique key that is generated when the category is registered.

Identifiers

Similar to categories, identifiers provide identification information, which allows businesses, services, and tModels to be associated with some identification scheme, such as model identification, or an industry group identification number.

❖ **Adding an identifier to a business, service, or tModel**

1 Expand the Publish folder.

2 Expand the Published Businesses, Services, or tModels folder.

3 Expand the business, service, or tModel for which you are adding an identifier.

4 Right-click the Identifiers folder and select Add Identifier.

5 Follow the Add Identifier Entity wizard to add an identifier. See Table 6-8 on page 90 for a description of the identifiers properties.

❖ **Deleting an identifier from a business, service, or tModel**

1 Expand the Publish folder.

2 Expand the Published Businesses, Services, or tModels folder.

3 Expand the business, service, or tModel to which the identifier you are deleting belongs.

4 Expand the Identifiers folder.

5 Right-click the identifier you want to delete and select Delete.

Table 6-8: Identifier properties

Tab	Property	Description
General	Identification scheme	Select the identification scheme to use with the Web service.
	Name	The name of the identification.
	Value	Each identifier has a corresponding value.
	Key	A unique key that is generated when the identifier is registered.
Summary		Displays a summary of your selections. Click Finish to create the identifier, or click Back to change selections.

Contacts

A contact (name, phone number, address) for a given business or business service.

❖ **Adding a contact to a business**

1 Expand the Publish folder.

2 Right-click the Published Businesses folder.

3 Right-click the business to which you are adding a contact and select Add Contact.

4 Follow the Add Contact wizard to add a contact. See Table 6-9 on page 91 for a description of the contact properties.

❖ **Deleting a contact from a business**

1 Expand the Publish folder.

2 Expand the Published Businesses folder.

3 Expand the business which contains the contact you are deleting.

4 Right-click the contact you want to delete and select Delete.

Table 6-9: Contact properties

Tab	Property	Description
General	Contact	The name of the contact; this could be a company or organization name.
	Description	A contact description.
	Contact person	A contact person.
	Address	The address of the contact.
	Phone number	The phone number of the contact.
Summary		Displays a summary of your selections. Click Finish to create the contact, or click Back to change selections.

Discovery URLs

A discovery URL is used to retrieve discovery documents for a specific instance of a business entity.

❖ **Adding a discovery URL to a business or tModel**

1 Expand the Publish folder.

2 Right-click the Published Businesses or Published tModel folder.

3 Right-click the business or tModel to which you are adding a discovery URL and select Add Discovery URL.

4 Follow the Add Discovery URL wizard to add a Discovery URL. See Table 6-10 on page 91 for a description of the Discovery URL properties.

❖ **Deleting a discovery URL from a business or tModel**

1 Expand the Publish folder.

2 Expand the Published Businesses or Published tModel folder.

3 Expand the business or tModel which contains the discovery URL you are deleting.

4 Right-click the discovery URL you want to delete and select Delete.

Table 6-10: Discovery URL properties

Tab	Property	Description
General	Discovery URL	URL to the discovery document.
	Description	A description of the discovery document.
	Use type	
	Language	The language name and description.

Tab	Property	Description
Summary		Displays a summary of your selections. Click Finish to create the discovery URL, or click Back to change selections.

CHAPTER 7

The Private UDDI Server

This chapter describes how to configure, start, and manage the private Sybase UDDI server. Once the server is running and connected to the registry, use the Web console to administer UDDI data. See Chapter 6, “Web Console—Registry Services.”

Introduction

A private UDDI server is an internal server that contains a UDDI registry, located behind a firewall that is isolated from the public network. Access to administrative features and registry data is secure, and not shared with other registries.

The private UDDI server adheres to the following UDDI specifications published at at http://www.uddi.org:

- API Specification 2.04 – the application user interface for accessing UDDI programmatically.
- Data Structure Reference 2.03 – the datatypes that make up the complete amount of information provided within the UDDI service description framework.
- Operator Specification 2.01 – general guidelines for managing and maintaining the registration information within the UDDI registry.

Installing and starting the private UDDI server

The EAServer installation program installs the private UDDI server by default, which installs the server and the scripts that start the server, and a database that contains registry information.

Note You must install JDK 1.4 to run the UDDI server. A typical EAServer installation does not include JDK 1.4.

❖ **Starting the private UDDI server in UNIX**

1 From the command line, verify that the JAGUAR environment variable is set to your EAServer installation directory.

2 Verify that EAServer is not running.

3 In the *$JAGUAR/bin* subdirectory, enter the command:

```
./uddiserver.sh
```

❖ **Starting the private UDDI server in Windows**

1 From the command line, verify that the JAGUAR environment variable is set to your EAServer installation directory.

2 Verify that EAServer is not running.

3 In the *%JAGUAR%\bin* subdirectory, enter the command:

```
uddiserver.bat
```

Starting and connecting to the private UDDI registry

EAServer contains a connection cache, UDDIServerCache, from which you can manage connection information. The UDDIServerCache uses the Adaptive Server Anywhere (ASA) database by default.

Starting the default UDDI registry

An ASA database is installed as the default UDDI server database.

❖ **Starting the ASA database in UNIX**

1 Edit the *$JAGUAR/bin/uddidb.sh* file. Modify the ASA_HOME variable to point to the location of your ASA installation.

2 From the command line in the *$JAGUAR/bin* subdirectory, enter:

```
./uddidb.sh
```

❖ **Starting the ASA database in Windows**

1 Edit the *%JAGUAR%\bin\uddidb.bat* file. Modify the ASA_HOME variable to point to the location of your ASA installation.

2 From the command line in the *%JAGUAR%\bin* subdirectory, enter:

```
uddidb.bat
```

Configuring other private UDDI registries

Along with the ASA database, the private UDDI server supports the Adaptive Server Enterprise database and Oracle databases as repositories for registry information. The SQL commands used to create the required database tables are in:

- *$JAGUAR/repository/WebApplication/uddi-server/META-INF/createdb_oracle.sql* (or the Windows equivalent) and
- *$JAGUAR/repository/WebApplication/uddi-server/META-INF/createdb_ase.sql* (or the Windows equivalent)

To use either of these databases with your private UDDI registry, use EAServer Manager to modify the UDDIServerCache connection cache to access the database of your choice (or create a new connection cache with the appropriate settings). See Chapter 4, “Database Access,” in the *EAServer System Administration Guide* for more information.

If you create a new connection cache, you can use the Web console to change to the connection cache:

1 Connect to the private UDDI server (UDDI on localhost).

2 Right-click the Registry Administration folder and select Configure Connection Cache.

3 Follow the wizard instructions to change the connection cache.

Connecting to the private UDDI registry

Before you can connect to the private UDDI registry, verify that you have:

1 Started the UDDI server.

2 Started the registry.

3 Started the Web console.

❖ **Connecting to the private UDDI registry**

1 From the Web console, right-click UDDI on localhost and select Connect.

2 The Web console connects to the UDDI registry using the default values:

- Query URL – http://localhost:8080/uddi-server/inquiry.
- Publish URL – http://localhost:8080/uddi-server/publish.
- Username – ADMIN.
- Password – ACCOUNT.

If you have modified the host name setting in your EAServer installation from localhost to the actual host name, you must make the same change in the Query URL and Publish URL fields above.

Managing the private UDDI

This section describes how to administer the private UDDI server from the Web console.

Once connected to the private UDDI server, you see three folders:

- Administration – contains the Registry and Security Administration folders, from which you can administer the private UDDI server. See “Administering the private UDDI” on page 97.
- Search – allows you to search the private UDDI repository for specific information based on your input. See “Inquiries and searches” on page 82.
- Publish – allows you to publish information about your business, Web service tModel, and so on, to the private UDDI repository. See “Publishing” on page 84.

Administering the private UDDI

Private UDDI administration consists of:

- Changing the connection cache – allows you to change connection caches, which allows you to use another database as the private UDDI repository.
- Initializing the database – this option is available from the Connection Cache property sheet, and appears only if the database being used as the repository has not been initialized.
- Managing user names, passwords, and permissions – user information is contained in the *$JAGUAR/Repository/WebApplication/uddi-server/WEB-INF/conf/juddi.user* file.

❖ Changing the connection cache

1. Connect to the private UDDI registry.
2. Right-click the Registry Administration folder and select Configure Connection Cache.
3. Follow the instructions in the Configure Connection Cache wizard to change the connection cache. You can select only a connection cache that is already defined in the EAServer to which you are connected.

❖ Initializing the repository database

If the database has not been initialized, follow this procedure to create the tables within the database that are required by the UDDI server.

1. Connect to the private UDDI registry.
2. Highlight the Registry Administration folder.

   You see the status of the database; true - the database is initialized, or false - the database has not been initialized.
3. If false, select Initialize Database.

   To start the initialized database run either *uddidb.bat* (Windows) or *uddidb.sh* (UNIX).

Managing user names, passwords, and access

The *$JAGUAR/Repository/WebApplication/uddi-server/WEB-INF/conf/juddi.user* XML file contains information about the users who have access to the private UDDI server. Edit this file and restart the server, to add authenticated users and passwords. You can provide users access to all features of the private UDDI server (including administration services) by setting the "admin" option to "true", as this sample *juddi.user* fragment illustrates:

```
<user userid="TEST_USER" password="SECRET" name="Test User"  admin="true" />
```

CHAPTER 8

Using wstool, wstkeytool, wstant, and wstkeytoolant

This chapter contains instructions on how to use wstool, either by itself, or with wstant.

Introduction

wstool is a command line interface that allows you to administer, monitor, and deploy Web services contained in the EAServer Web service container.

You can use wstool from the command line, from scripts or makefiles, or with Jakarta Ant (wstant).

wstkeytool is a command line interface that allows you to manage a keystore (database or file) of private keys and their associated X.509 certificate chains. In this case, the keystore is the EAServer security module. wstkeytool commands allow you to manipulate the EAServer security module without having to use Security Manager.

See the *EAServer Security Administration and Programming Guide* for definitions and information about Security Manager.

You can use wstkeytool from the command line, from scripts or makefiles, or with Jakarta Ant (wstkeytoolant).

Working with wstool and wstkeytool

Before using wstool or wstkeytool, make sure that the JAGUAR environment variable is set to the EAServer installation directory. Use the following scripts to run wstool and wstkeytool:

- **UNIX** *$JAGUAR/bin/wstool* and *$JAGUAR/bin/wstkeytool*
- **Windows** *%JAGUAR%\bin\wstool.bat* and *%JAGUAR%\bin\wstkeytool.bat*

wstool and wstkeytool syntax

The syntax for wstool and wstkeytool is:

wstool or wstkeytool [*connection-arguments*] [*command*]

Where:

- *connection-arguments* specify optional parameters required to connect to the server, including:

Flag	To specify
`-h` or `-host`	Server host name; default is the value of JAGUAR_HOST_NAME (localhost)
`-n` or `-port`	Web services host port number; default is 8080
`-u` or `-user`	User name; default is “jagadmin”
`-p` or `-password`	Password; default is “” (no password)
`-k` or `-protocol`	Communication protocol; default is “http”
`-l` or `-logfile`	Log file name; default is “System.out”
`-r` or `-pin` (wstkeytool only)	Keystore PIN of EAServer’s PKCS#11 token; default is “sybase”
`-s` or `-provider` (wstkeytool only)	The keytool provider. This option is not used with EAServer.

- *command* is a wstool or wstkeytool command.

For example, to connect to the server running on “paloma” at HTTP port “9005”, using account “jagadmin” with password “secret” enter:

wstool -h paloma -n 9005 -p secret or wstkeytool -h paloma -n 9005 -p secret

You can omit the *-u* flag because *jagadmin* is the default user name.

Note wstool and wstkeytool command options are not case sensitive.

Return codes

wstool and wstkeytool commands return the following codes:

0 – if the command runs successfully, and the result is true/success

1 – if the command runs successfully, and the result is false/failure

2 – if an exception is thrown during the running of the command

Help

You can display usage for any wstool or wstkeytool command by using the help option. For example to display all of the wstool or wstkeytool commands, enter:

`wstool help` or `wstkeytool help`

You can also display individual command help. For example, to display options and valid usage for the wstool delete command, enter:

```
wstool help delete
```

To display options and valid usage for the wstkeytool deleteCert command, enter:

```
wstkeytool help deleteCert
```

Verbose

All wstool and wstkeytool commands include the verbose option, which displays stack trace information, if any is generated, when you run the command. The default value is false. For example, to display stack trace information for the wstool delete command, enter:

```
wstool delete -verbose true
Service:CollectionName/WebServiceName
```

Entity identifiers

Many wstool and wstkeytool commands take one or more entity identifiers as arguments. An entity identifier is a string of the form *EntityType:EntityName* that uniquely identifies an entry in the repository; for example, a server, component, collection, or keystore name.

Table 8-1 provides examples of entity identifiers for each entity type.

Table 8-1: Example entity identifiers

Entity identifier	Specifies
`component:SVU/SVULogin` (wstool)	Component named SVULogin that is installed in the SVU package. The package name is included because EAServer components always reside in packages.
`collection:MyCollection` (wstool)	The Web services collection named MyCollection.
`method:SVU/SVULogin/isLogin` (wstool)	The isLogin method of component SVULogin in package SVU.
`role:MyRole` (wstool)	The role named MyRole.
`server:Jaguar` (wstool)	The server named Jaguar.
`service:MyWcoll/MyWebService` (wstool)	The Web service named MyWebService contained in the MyWcoll Web collection.
`methodParams:SVU/SVULogin/isLogin` (wstool)	The method parameters for the isLogin method of component SVULogin in package SVU.
`Key:name` (wstkeytool)	The name of a private key contained in the keystore.

Not all wstool or wstkeytool commands support every type of entity in the repository. Use the help option to see the supported entities for each command.

When a command specifies an invalid entity type, an error message displays.

Working with wstant and wstkeytoolant

wstant and wstkeytoolant lets you run wstool and wstkeytool commands from Ant build files. This allows you to write build files that automate many development, deployment, and management 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 functions are similar to other build tools (such as *make*, *gnumake*, or *jam*) but are 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 makefile) are written in XML. Like makefiles, 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.

wstant and wstkeytoolant scripts requires a full JDK installation. If you are running wstant or wstkeytoolant from an EAServer client install, make sure you have installed the full JDK. By default, only the JRE files are installed.

Before running wstant or wstkeytoolant, verify that:

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

  By default, wstant and wstkeytoolant searches for Jakarta Ant in *%JAGUAR%\jakarta-ant* (Windows) or *$JAGUAR/jakarta-ant* (UNIX). If you install Jakarta Ant in a different location, set the ANT_HOME environment variable to reflect the change before you run wstant or wstkeytoolant scripts.

  You can also set ANT_HOME in the user environment file, *%JAGUAR%\bin\user_setenv.bat* (Windows) or *$JAGUAR/bin/user_setenv.sh* (UNIX). wstant and wstkeytoolant scripts check the user environment file each time it runs.

wstant and wstkeytoolant scripts

The following scripts are provided for running Ant with wstool and wstkeytool commands:

- **Windows** *%JAGUAR%\bin\wstant.bat* and *%JAGUAR%\bin\wstkeytoolant.bat*
- **UNIX** *$JAGUAR/bin/wstant* and *$JAGUAR/bin/wstkeytoolant*

wstant and wstkeytoolant syntax

wstant and wstkeytoolant scripts uses this syntax:

wstant or wstkeytoolant *[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, which lets you specify a build file other than the default *build.xml* for the Ant XML build file.

wstant sample files

The EAServer installation includes wstant sample files. The *ReadmeForWSTant.html* and *build.xml* files are located in *%JAGUAR%\sample\wst_samples\JavaClass\SimpleJavaClass* (Windows) or *$JAGUAR/sample/wst_samples/JavaClass/SimpleJavaClass* (UNIX). See the *ReadmeForWSTant.html* file for instructions on using the samples.

wstool commands

Description

This section contains information on wstool commands, and lists the commands that wstool accepts.

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. wstool commands are divided into four sections:

- UDDI administration commands on page 105
- Server management commands on page 109

- Web service administration commands on page 115
- Security commands on page 141

UDDI administration commands

Description

UDDI commands allow you to publish and unpublish Web service information to and from a UDDI registry.

Command list

Table 8-2 lists the UDDI administration commands described in this section.

Table 8-2: wstool UDDI administration commands

command name	Description
inquiry	Queries a UDDI registry for business, service, or tModel information.
publish	Publishes Web service information to a UDDI registry.
unpublish	Unpublishes Web service information from a UDDI.

inquiry

Description

Queries a UDDI registry for business, service, or tModel information.

Syntax

Command line:

inquiry
[-inquiryURL *URL*]
[-business *business_name*]
[-exact *true* | *false*]
[-service *service_name*]
[-tmodel *tModel_name*]

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="inquiry" > <wst_antTask command="inquiry"
[inquiryURL="*URL"*]
[business= "*business_name"*]
[exact="*true* | *false*"]
[service="*service_name"*]
[tmodel="*tModel_name" /*>

Where:

Option	Description
inquiryURL	Inquiry URL used to connect to the registry. Required.
business	Provide the business name if querying a business. Provide a business key if querying a service, which lists only those services for the particular business. If the key is not specified, all the services that match all business are listed.
exact	True or false. If true (the default), only entities with exact matches are listed. If false, all entities that begin with the business, service, or tModel name specified are listed.
service	Specify the service name to query a service.
tmodel	Specify the tModel name to query a tModel.

Examples

This command queries information about “myBusiness” from the IBM test registry:

```
wstool inquiry -inquiryURL http://uddi.ibm.com/testregistry/inquiryapi
-business myBusiness
```

Ant build example:

```
<wst_antTask command="inquiry"
inquiryURL="https://uddi.ibm.com/testregistry/inquiryapi"
business="myBusiness"/>
```

publish

Description

Publishes Web service information to a UDDI registry.

Syntax

Command line:

publish
[-inquiryURL *URL*]
[-publishURL *URL*]
[-user *user_name*]
[-business *business_name*]
[-pass *password*]
[-serviceURL *URL*]
[-publishName *name*]
[-tmodel *tModel_name*]

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="publish" > <wst_antTask command="publish"
[inquiryURL="*URL"*]
[publishURL=*"URL"*]
[user=*"user_name"*]
[business= "*business_name"*]
[pass="*password*"]
[serviceURL=*"URL"*]
[publishName=*"name"*]
[tmodel="*tModel_name" /*>

Where:

Option	Description
inquiryURL	Inquiry URL used to connect to the registry. Required.
publishURL	Publish URL used to connect to the registry. Required.
user	User name used to connect to the UDDI registry URL. Required.
business	Provide the business name if publishing a business or specify the business key if publishing a service.
pass	The password used to connect to the UDDI registry URL.
serviceURL	The service URL of the service to be published.
publishName	Specifies a name with which the tModel can be published. to publish a service or a tModel, you must specify the publish.
tmodel	Specifies the tModel key that associates the service to a specific tModel.

Examples

This command publishes information about "testservice" to the IBM test registry:

```
wstool publish -inquiryURL http://uddi.ibm.com/testregistry/inquiryapi
-publishURL https://uddi.ibm.com/testregistry/publishapi -user testuser
-business 6B9DD2D0-D81E-11D7-A0BA-000629DC0A13 -pass secret -serviceURL
http://webservicehost:8080/ws/services/testservice -publishName
testpublish -tmodel 216DD2D0-A21E
```

Ant build example:

```
<wst_antTask command="publish"
inquiryURL="https://uddi.ibm.com/testregistry/inquiryapi"
publishURL="https://uddi.ibm.com/testregistry/publishapi" user="me"
pass="secret" business="myTestBusinessOnly"/>
```

unpublish

Description

Unpublishes Web service information from a UDDI registry.

Syntax

Command line:

unpublish
[-inquiryURL *URL*]
[-publishURL *URL*]
[-user *user_name*]
[-business *business_name*]
[-pass *password*]
[-serviceURL *URL*]
[-serviceKey *key*]
[-tmodel *true*]

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="unpublish" > <wst_antTask command="unpublish"
[inquiryURL=*"URL"*]
[publishURL=*"URL"*]
[user=*"user_name"*]
[business= *"business_name"*]
[pass=*"password"*]
[serviceURL=*"URL"*]
[serviceKey=*"key"*]
[tmodel=*"tModel_name"* />

Where:

Option	Description
inquiryURL	Inquiry URL used to connect to the registry. Required.
publishURL	Publish URL used to connect to the registry. Required.
user	User name used to connect to the UDDI registry URL. Required.
business	Provide the business name if unpublishing a business or specify the business key if unpublishing a service.
pass	The password used to connect to the UDDI registry URL.
serviceURL	The service URL of the service being unpublished.
serviceKey	You must specify a service key to unpublish a tModel.
tmodel	Specifies the tModel key that associates the service to a specific tModel.

Examples

This command unpublishes information regarding “testservice” from the IBM test registry:

```
wstool unpublish -inquiryURL http://uddi.ibm.com/testregistry/inquiryapi
```

```
-publishURL https://uddi.ibm.com/testregistry/publishapi -user testuser
-business 6B9DD2D0-D81E-11D7-A0BA-000629DC0A13 -pass secret -serviceURL
http://webservicehost:8080/ws/services/testservice -serviceKey 1234 -tmodel
216DD2D0-A21E
```

Ant build example:

```
<wst_antTask command="unpublish"
inquiryURL="https://uddi.ibm.com/testregistry/inquiryapi"
publishURL="https://uddi.ibm.com/testregistry/publishapi" user="me"
pass="secret" business="myTestBusinessOnly"/>
```

Server management commands

Description

Server management commands allow you to start, stop, and manage the server, as well as manage listeners for EAServer.

Command list

Table 8-3 lists the server management commands.

Table 8-3: wstool server management commands

command name	Description
list	Lists entities in the repository.
refresh	Refreshes a server or Web service collection.
restart	Restarts the server to which you are connected.
shutdown	Shuts down the server to which you are connected.

list

Description

Returns a list of entities from the server's repository, depending on the type of entity entered.

Note Entity type is not an option, do not use a "-" when specifying an entity type.

Syntax

Command line:

list
[Collections]
[CompType]

[Components]
[Listeners]
[Methods]
[Packages]
[Params]
[Props]
[PropsValue]
[ReturnType]
[Roles]
[ServerProps]
[ServerVersion]
[ServiceName]
[Services]
[Stats]
[URL]
[WSDD]
[WSDL]
[typemappings]
[undefTypes]
Entity

Ant build file:

```
<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="list" > <wst_antTask command="list"
[option="option_depending_on_entity"] >
```

Where:

Type	Description
Collections	Returns a list of Web service collections.
CompType	Returns the component type. Entity is in the form of component:*PackageName/ComponentName*.
Components	Returns a list of SOAPable components available on the server.
Listeners	Returns a list of listeners in the format of "<protocol>:<host>:<port>". For example, "http:localhost:8080"

Type	Description
Methods	Returns a list of methods for the entity. Entity can be in the form of either: • service:*CollectionName/ServiceName* or • component:*PackageName/ComponentName* Include the `-methodType` option and specify the type of methods returned: `allowed` – list only the allowed methods. `disallowed` – list only the disallowed methods. `all` – list all methods (default).
Packages	Returns a list of SOAPable packages available on the server.
Params	Returns a list of parameters for a given method. Entity is in the format of: method:*CollectionName/ServiceName/MethodName*
Props	Returns a list of properties of a given entity, for example: collection:*CollectionName*
PropsValue	Returns the property value for the given property. Use the `-name` argument and provide the name of the property for which the value is returned. Entity can be one of: • collection:*CollectionName* • server:*ServerName*
ReturnType	Returns the return type of a given method. Entity is in the form of: method:*CollectionName/ServiceName/MethodName*
Roles	Returns a list of roles for a given Web service or Web service method. Entity is one of: • method:*CollectionName/ServiceName/MethodName* • service:*CollectionName/ServiceName*
ServerProps	Returns a list of server properties.
ServerVersion	Returns the server version.
ServiceName	Returns the Web service name of a given component. Entity is in the form of: component:*PackageName/ComponentName*

Type	Description
Services	Returns the list of Web services for a given collection. Use the `-serviceType` argument with one of the following options: all – list all Web services active – list only active Web services Entity is in the form of: collection:*CollectionName*
Stats	Displays the statistics for a given Web service or Web service method. Entity is one of: • method:*CollectionName/ServiceName/MethodName* • service:*CollectionName/ServiceName* **Note** You must first enable statistics monitoring before you can display statistics. See startStats on page 132.
URL	Returns the service URL of a given Web service is . Entity is in the form of: service:*CollectionName/ServiceName*
WSDD	Lists the *.wsdd* of a given Web service. Use the `-out` argument and supply a file name to direct the *.wsdd* to a file. The default file is *collectionName_serviceName.wsdd*. Entity is in the form of: service:*CollectionName/ServiceName*
WSDL	Lists the *.wsdl* of a given Web service. Use the `-out` argument and supply a file name to direct the *.wsdl* to a file. The default file is *collectionName_serviceName.wsdl*. Entity is in the form of: service:*CollectionName/ServiceNam*e
typemappings	Returns a list of the type mappings for a given Web service. Entity is in the format of: service:*CollectionName/ServiceName*
undefTypes	Returns a list of the undefined types for a given soapable component. Entity is on of: • method:*PackageName/ComponentName/MethodName* • class name
Entity	Varies depending on the selected option.

Examples **Example 1** This command lists all the listeners running on the server:

```
wstool list Listeners
```

Example 2 This command directs the WSDL for MyWebService to the *test.wsdl* file:

```
wstool list wsdl -out test.wsdl service:MyCollection/MyWebService
```

Ant build example:

```
<wst_antTask command="list" type="wsdl" entity:
"service:MyCollection/MyWebService"/>
```

refresh

Description

Refreshes a server or Web service collection, depending on the entity. Also refreshes the child properties of the specified entity. For example, if you refresh a server, all the server properties that belong to the server are refreshed.

Syntax

Command line:

refresh
entity

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="refresh" > <wst_antTask command="refresh"
entity="*entity*" />

Where:

Option	Description
entity	Can be one of: • server:*ServerName* – identifies the server you are refreshing. • collection:*WebServiceCollectionName* – identifies the Web service collection you are refreshing.

Examples

This command refreshes the EAServer named "Jaguar:"

```
wstool refresh server:Jaguar
```

Ant build example:

```
<wst_antTask command="refresh"
entity="server:Jaguar"/>
```

restart

Description — Restarts the server to which you are connected.

Syntax — **Command line:**

```
restart
```

Ant build file:

```
<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="restart" > <wst_antTask command="restart"
```

Examples — This command restarts the server to which you are connected:

```
wstool restart
```

Ant build example:

```
<wst_antTask command="restart" />
```

shutdown

Description — Shuts down the server to which you are connected.

Syntax — **Command line:**

```
shutdown
```

Ant build file:

```
<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="shutdown" > <wst_antTask command="shutdown"
```

Examples — This command shuts down the server to which you are connected:

```
wstool shutdown
```

Ant build example:

```
<wst_antTask command="shutdown" />
```

Web service administration commands

Description — Web service administration commands allow you to manage most aspects of Web services.

Command list — Table 8-4 lists the Web service administration commands.

Table 8-4: wstool Web service commands

command name	Description
activate	Activates a Web service and makes it available to clients.
allowMethods	Makes available to clients the selected methods of a Web service.
deactivate	Deactivates a Web service and makes it unavailable.
delete (1)	Deletes a Web service.
delete (2)	Deletes a Web service collection.
deploy (1)	Creates and deploys a Web service from the implementation class file.
deploy (2)	Creates and deploys a Web service from a JAR file.
deploy (3)	Creates and deploys a Web service collection from a WAR file.
disallowMethods	Makes Web service methods unavailable to Web service clients.
export	Exports a Web service collection to a Sybase Web services WAR file.
exposeComponent	Exposes an EAServer component as a Web service.
getTMjar	Creates a type mapping JAR file.
isActive	Returns a message that a given Web service is either “active” or “inactive.”
isAllowed	Checks if the method is available to a client as a Web service endpoint.
isStatsEnabled	Determines if statistic logging for a given Web service is enabled or not.
refresh	Refreshes a server or Web service collection.
resetStats	Resets the runtime monitor statistics of a given Web service collection.
set_props	Sets the value of the property for a component, Web application, or a Web service.
startStats	The runtime monitor starts and monitors statistics of a given Web service collection or Web service.
stopStats	The runtime monitor stops monitoring statistics of a given Web service collection or Web service.
wsdl2Java	Generates client artifacts and a client template capable of accessing server-side Web services.
java2Wsdl	Generates a WSDL file from the Java implementation file.

activate

Description

Activates a Web service in a given Web service collection so that it is available to clients.

Syntax

Command line:

activate
entity

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="activate" > <wst_antTask command="activate"
entity="*entity*" >

Where:

Option	Description
entity	Service:*CollectionName/ServiceName* – identifies the Web service you are activating.

Examples

This command activates MyWebService which is contained in MyCollection:

```
wstool activate Service:MyCollection/MyWebService
```

Ant build example:

```
<wst_antTask command="activate"
entity="service:myCollection/myService"/>
```

allowMethods

Description

Makes Web service methods available to clients.

Syntax

Command line:

allowMethods
entity

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="allowmethods" > <wst_antTask
command="allowmethods"
entity="*entity*" >

Where:

Option	Description
entity	method:*CollectionName/ServiceName/m1, m2, m3* – identifies the Web service to which the methods being made available belong, and a comma-separated list of method names that are available to a client. The entity must be in quotes.

Examples

This command makes testmethod1 and testmethod2 available to a Web service client that belongs to MyWebService:

```
wstool allowMethods "method:WebColl/MyWebService/testmethod1, testmethod2"
```

Ant build example:

```
<wst_antTask command="allowMethods"
entity="method:myCollection/myService/myMethod"/>
```

deactivate

Description

Deactivates a Web service so that it is unavailable to clients.

Syntax

Command line:

deactivate
entity

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="deactivate" > <wst_antTask command="deactivate"
entity="*entity*" >

Where:

Option	Description
entity	service:*CollectionName/ServiceName* – identifies the Web service you are deactivating.

Examples

This command deactivates MyWebService which is contained in MyCollection:

```
wstool deactivate service:MyCollection/MyWebService
```

Ant build example:

```
<wst_antTask command="deactivate"
entity="service:myCollection/myService"/>
```

delete (1)

Description

Deletes a Web service from a given Web service collection. The service element in the *server-config.wsdd* file is deleted and the files indicated by the "files" parameter of that service element are also deleted.

Syntax

Command line:

delete
entity

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="delete" > <wst_antTask command="delete"
entity="*entity*" >

Where:

Option	Description
entity	Service:*CollectionName/ServiceName* – identifies the Web service you are deleting.

Examples

This command deletes MyWebService:

```
wstool delete Service:MyWebCollection/MyWebService
```

Ant build example:

```
<wst_antTask command="delete"
entity="service:myCollection/myService"/>
```

delete (2)

Description

Deletes a Web service collection.

Syntax

Command line:

delete
entity

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="delete" > <wst_antTask command="delete"
entity="*entity*" >

Where:

Option	Description
entity	collection:*CollectionName* – identifies the Web service collection you are deleting.

Examples

This command deletes MyWebServiceCollection:

```
wstool delete collection:MyWebServiceCollection
```

Ant build example:

```
<wst_antTask command="delete"
entity="collection:myCollection/>
```

deploy (1)

Description

Creates and deploys a Web service using an implementation class file. This command creates a Web service in the Web service collection name provided by you, or uses "ws" as the default. This command creates the Web service collection if it does not already exist.

Syntax

Command line:

deploy
[-overwrite *true* | *false*]
[-collection *collectionName*]
[-include *directory*]
[-classpath *path*]
entity

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="deploy" > <wst_antTask command="deploy"
[overwrite=*"true* | *false"*]
[collection=*"collectionName"*]
[include=*"directory"*]
[classpath= *"path"*]
entity =*"className"* >

Where:

Option	Description
overwrite	If set to true, overwrites an existing Web service if it has the same service name. The default is false.
collection	Specifies the collection name. `ws` is the default Web collection.
include	Specifies the directory that contains any dependent classes. For example: *d:\foo* This option must be in quotes.
classpath	Specifies additional JARs/classes to set in classpath. **Note** JARs must be specified in quotes.
entity	The file that you are deploying. *entity* should be located in the current directory, or provide the full path. If deploying from an implementation class file, *entity* is in the format of *foo.bar.myclass* or *foo.bar.myclass.class*.

Examples

This example deploys the Web service from the com.sybase.mytest class file to MyServiceCollection:

```
wstool deploy -overwrite true -collection MyServiceCollection -include
"d:\classes;d:\moreclasses" com.sybase.mytest
```

Ant build example:

```
<wst_antTask command="deploy"
collection="CollectionName"
include="d:\moreclasses"
entity="com.sybase.myTest"/>
```

Note You cannot deploy a class that uses "DefaultNamespace" as the package name. For example:
`wstool deploy -include "d:\mytest" DefaultNamespace.myTest` is not valid.

deploy (2)

Description

Creates and deploys a Web service from a Sybase Web services JAR file.

Syntax

Command line:

deploy
[-overwrite *true* | *false*]
[-collection *collectionName*]
[-include *directory*]
[-classpath *path*]
entity

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="deploy" > <wst_antTask command="deploy"
[overwrite=*"true | false"*]
[collection=*"collectionName"*]
[include=*"directory"*]
[classpath= *"path"*]
entity =*"file"* >

Where:

Option	Description
overwrite	If set to true, overwrites an existing Web service if it has the same service name. The default is false.
collection	Specifies the collection name, if you are deploying a JAR file . `ws` is the default Web collection.
include	Specifies the directory that contains any dependent classes. For example: *d:\foo* This option must be in quotes.
classpath	Specifies additional JARs/classes to set in classpath. **Note** JARs must be specified in quotes.
entity	The file that you are deploying. *entity* should be located in the current directory, or provide the full path.

Examples

This example deploys the Web service contained in the *MyWebService.jar* file:

```
wstool deploy MyWebService.jar
```

Ant build example:

```
<wst_antTask command="deploy"
entity="d:\wstool\test\deploy\service.jar"/>
```

deploy (3)

Description

Creates and deploys a Web service collection from a Sybase Web services WAR file.

Syntax

Command line:

deploy
[-overwrite *true* | *false*]
[-include *directory*]
[-classpath *path*]
entity

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="deploy" > <wst_antTask command="deploy"
[overwrite=*"true | false"*]
[include=*"directory"*]
[classpath= *"path"*]
entity =*"file"* >

Where:

Option	Description
overwrite	If set to true, overwrites an existing Web service collection if it has the same collection name. The default is false.
include	Specifies the directory that contains any dependent classes. For example: *d:\foo* This option must be in quotes.
classpath	Specifies additional JARs/classes to set in classpath. **Note** JARs must be specified in quotes.
entity	The file that you are deploying. *entity* should be located in the current directory, or provide the full path.

Examples

This example deploys the Web service collection contained in the *MyWebServiceCollection.war* file:

```
wstool deploy MyWebServiceCollection.war
```

Ant build example:

```
<wst_antTask command="deploy"
entity="d:\wstool\test\deploy\collection.war"/>
```

disallowMethods

Description — Makes the listed methods unavailable to a Web service client.

Syntax

Command line:

```
disallowMethods
entity
```

Ant build file:

```
<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="disallowMethods" > <wst_antTask
command="disallowMethods"
entity="entity" >
```

Where:

Option	Description
entity	method:*CollectionName/ServiceName/m1, m2* – identifies the Web service and a comma-separated list of methods you are making unavailable. Entity must be specified in quotes.

Examples — This command makes MyMethod1 and MyMethod2 unavailable to clients:

```
wstool disallowMethods "method:MyWebCollection/MyWebService/Mymethod1,
MyMethod2"
```

Ant build example:

```
<wst_antTask command="disallowMethods"
entity="method:myCollection/myService/myMethod"/>
```

export

Description — Exports a Web service collection to a Sybase Web archive (WAR) file.

Syntax

Command line:

```
export
[-out outputFile]
[-component true | false]
[-configXML true | false]
entity
```

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="export" > <wst_antTask command="export"
[out=*"outputfile"*]
[component=*"true* | *false"*]
[configXML=*"true* | *false"*]
entity =*"entity"* >

Where:

Option	Description
out	Specifies the exported archive file. Provide the complete path or the file is placed in the current directory. For a Web service collection the output is in the format of *collectionName.war*.
component	Set this option to true (default) if the service contained in the collection is exposed as a Jaguar component, which will export the Jaguar component related files.
configXML	Set this option to true (default) to export the *sybase_easerver_config.xml* file.
entity	collection:*CollectionName* – identifies the Web service collection you are exporting.

Examples

This command exports MyWebServiceCollection to the *MyWebServiceCollection.war* file:

```
wstool export -out MyWebServiceCollection.war collection:MyWebCollection
```

Ant build example:

```
<wst_antTask command="export"  out="outDir"
entity="collection:myCollection"/>
```

exposeComponent

Description

Exposes an EAServer component as a Web service.

Syntax

Command line:

exposeComponent
[-collection *webCollection*]
[-service *webService*]
[-tm *typeMapping*]
[-tmJar *jarFile*]
entity

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="exposeComponent" > <wst_antTask
command="exposeComponent"
[collection=*"webCollection"*]
[service=*"webService"*]
[tm=*"typeMapping"*]
[tmJar=*"jarFile"*]
entity =*"package/component"* >

Where:

Option	Description
collection	Specifies the name of the Web service collection, to which the Web service belongs. `ws` is the default.
service	Specifies the Web service name to which the component is exposed to. The default is *PackageName_ComponentName*.
tm	Specifies the type mapping file name for any undefined custom datatypes.
tmJar	Specifies the full path to the JAR file that contains any custom datatype mappings required by the component.
entity	The name of the EAServer package/component being exposed.

Examples

This command exposes myPkg/myComp as a Web service:

```
wstool exposeComponent -tm myTM.map -tmJar myTM.jar myPkg/myComp
```

Ant build example:

```
<wst_antTask command="exposeComponent"
entity="component:myPackage/myComponent"/>
```

getTMjar

Description

Creates a JAR file that contains type mappings identified by the class option and associates it with an entity for which the type mapping is needed.

Syntax

Command line:

getTMjar
[-class *classname*]
[-outJar *jarFile*]

[-overwrite *true* | *false*]
entity

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="getTMjar" > <wst_antTask command="getTMjar"
[class=*"classname"*]
[outjar=*"jarFile"*]
[overwrite=*"true* | *false"*]
entity =*"entity"* >

Where:

Option	Description
class	The name of the class for which the type mapping JAR is needed.
outJar	The name of the JAR to be used for the output of the class. The default is *className.jar*
overwrite	overwrites the JAR, if it already exists. The default is not to overwrite.
entity	Service:*CollectionName/ServiceName* – identifies the Web service that requires the type mappings contained in the JAR.

Examples

This command creates a *testclass.jar* file that contains the type mappings contained in testclass and required by MyWebService:

```
wstool getTMjar -class testclass -outjar testclass.jar
Service:MyWebServiceCollection/MyWebService
```

Ant build example:

```
<wst_antTask command="getTMjar"
class="myPkg.mysampleClass"
entity="service:myCollection/myService"/>
```

isActive

Description

Returns a message that a given Web service is either “active” or “inactive.”

Syntax

Command line:

isActive
entity

Ant build file:

```
<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="isActive" > <wst_antTask command="isActive"
entity="entity" >
```

Where:

Option	Description
entity	Service:*CollectionName/ServiceName* – identifies the Web service which is either "active" or "inactive."

Examples

This command returns either "active" or "inactive" for MyWebService:

```
wstool isActive Service:MyWebServiceCollection/MyWebService
```

Ant build example:

```
<wst_antTask command="isactive"
entity="service:myCollection/myService"/>
```

isAllowed

Description

Checks if the method is available to a client as a Web service endpoint.

To make methods available to clients, see allowMethods on page 117.

Syntax

Command line:

```
isAllowed
entity
```

Ant build file:

```
<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="isAllowed" > <wst_antTask command="isAllowed"
entity="entity" >
```

Where:

Option	Description
entity	method:*CollectionName/ServiceName/MethodName* – the name of the method being queried.

Examples

This command checks to see if MyMethod is available to the client:

```
wstool isAllowed method:MyWebServiceCollection/MyWebService/MyMethod
```

Ant build example:

```
<wst_antTask command="isallowed"
entity="method:myCollection/myService/myMethod"/>
```

isStatsEnabled

Description

Determines if statistic logging for a given Web service or Web service collection is enabled or not. Returns true if enabled and false if disabled.

Syntax

Command line:

isStatsEnabled
entity

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="isStatsEnabled" > <wst_antTask
command="isStatsEnabled"
entity="*entity*" >

Where:

Option	Description
entity	The name of the Web service or Web service collection being queried: • service:*CollectionName/ServiceName* • collection:*CollectionName*

Examples

This command returns true if statistics are being gathered for MyWebService:

```
wstool isStatsEnabled service:MyWebCollection/MyWebService
```

Ant build example:

```
<wst_antTask command="isstatsenabled"
entity="service:myCollection/myService"/>
```

refresh

Description

Refreshes a server or Web service collection.

Syntax

Command line:

refresh
entity

Ant build file:

```
<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>

<!-- Refresh a collection on the server -->
   <target name="refresh" >
      <wst_antTask command="refresh"
entity="entity"/>
```

Where:

Option	Description
entity	Can be one of: • server:*ServerName* – identifies the server being refreshed. • collection:*CollectionName* – identifies the Web service collection being refreshed.

Examples

This example refreshes MyWebServiceColl, including all the Web services it contains.

```
wstool refresh collection:MyWebServiceColl
```

Ant build example:

```
<wst_antTask command="refresh" entity="collection:myCollection"/>
```

resetStats

Description

Resets the runtime monitor statistics of a given Web service collection or Web service.

Syntax

Command line:

resetStats
entity

Ant build file:

```
<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
```

<target name="resetStats" > <wst_antTask command="resetStats" entity="*entity*" >

Where:

Option	Description
entity	Can be one of: • service:*CollectionName/ServiceName* – identifies the Web service for which the statistics are reset. • collection:*CollectionName* – identifies the Web service collection for which the statistics are reset.

Examples

This command resets the runtime monitor statistics of MyWebServiceCollection:

```
wstool resetStats collection:MyWebServiceCollection
```

Ant build example:

```
<wst_antTask command="resetstats"
entity="service:myCollection/myService"/>
```

set_props

Description

Sets the value of the property for a Web service collection either using a name value pair or by specifying a file that contains the property name-value pair.

Syntax

Command line:

set_props
[entity *name value* | *file*]

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="set_props" > <wst_antTask command="set_props"
entity="*entity*" name="*nameOfProperty*" value="*propertyValue*">

Where:

Option	Description
entity	The entity that is being modified: collection:*CollectionName* – identifies the Web service collection for which the properties are set.

Option	Description
name	The name of the property being modified.
value	The new value of the property.
file	The name of the file that contains the name value pairs of properties being modified.

Examples

This command sets the description of MyWebServiceCollection:

```
wstool set_props collection:MyWebServiceCollection
com.sybase.jaguar.webApplication.description “My test description”
```

Ant build example:

```
<wst_antTask command="set_props"
entity="collection:myCollection"
name=”com.sybase.jaguar.webApplication.description”
value=”My test description” />
```

startStats

Description

The runtime monitor starts and monitors statistics of a given Web service collection or Web service.

Syntax

Command line:

startStats
entity

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="startStats" > <wst_antTask command="startStats"
entity=”*entity*” >

Where:

Option	Description
entity	The entity can be either: • service:*CollectionName/ServiceName* – identifies the Web service for which statistics are monitored. • collection:*CollectionName* – identifies the Web service collection for which statistics are monitored.

Examples

This command starts the runtime monitor and monitors statistics of MyWebServiceCollection:

```
wstool startStats collection:MyWebServiceCollection
```

Ant build example:

```
<wst_antTask command="startstats"
entity="service:myCollection/myService"/>
```

stopStats

Description

Stops the runtime monitor from monitoring statistics for a given Web service collection or Web service.

Syntax

Command line:

stopStats
entity

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="stopStats" > <wst_antTask command="stopStats"
entity="*entity*" >

Where:

Option	Description
entity	The entity can be either: • service:*CollectionName/ServiceName* – identifies the Web service for which statistics are no longer monitored. • collection:*CollectionName* – identifies the Web service collection for which statistics are no longer monitored.

Examples

This command stops monitoring statistics of MyWebServiceCollection:

```
wstool stopStats collection:MyWebServiceCollection
```

Ant build example:

```
<wst_antTask command="stopstats"
entity="service:myCollection/myService"/>
```

upgrade

Description

Upgrades all the Web services from a Web Services Toolkit version 4.*x* to 5.0 on the server to which you are connected.

Syntax

Command line:

upgrade

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="upgrade" > <wst_antTask command="upgrade">

Examples

This command upgrades all 4.x Web services to 5.0.

```
wstool upgrade
```

Ant build example:

```
<wst_antTask command="upgrade" />
```

wsdl2Java

Description

Generates Java code for client side artifacts from the WSDL, where WSDL URI is the URI (universal resource identifier) of the WSDL file.

wsdl2java generates a service implementation template file with a *.java.new* extension. Remove the *.new* extension and enter your business logic into the implementation file before deploying it as a Web service.

Note When you expose a component that uses EAServer-specific holder types as a Web service, the convention for generating the client-side holders classes is that they are always generated under a package.holders.type hierarchy. For example, when you expose a component as a Web service that uses BCD.MoneyHolder, the conversion on the client-side results in a JAX-RPC specific holder contained under BCD.holders.MoneyHolder. You will not use EAServer-specific types on the Web service client side.

Syntax

Command line:

wsdl2java
[-classpath *path*]
[-compile *true* | *false*]
[-factory *class_name*]

[-fileNS2pkg *file_name*]
[-genAll *true* | *false*]
[-genHelper *true* | *false*]
[-genImplTemplate *true* | *false*]
[-genRefrencedOnly *true* | *false*]
[-genSkeleton *true* | *false*]
[-genStub *true* | *false*]
[-gentestCase *true* | *false*]
[-gentypes *true* | *false*]
[-genWSDD *true* | *false*]
[-handlerFile *fileName*]
[-noImport *true* | *false*]
[-noWrapped *true* | *false*]
[-ns2pkg *package=namespace*]
[-outputDir *path*]
[-package *packageName*]
[-passwd *password*]
[-scope *Request* | *Application* | *Session*]
[-serverside *true* | *false*]
[-timeout *seconds*]
[-tm *argument*]
[-typeMappingVer *1.1* | *1.2*]
[-user *userName*]
WSDLURI

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="wsdl2java" > <wst_antTask command="wsdl2java"
[classpath=*"path"*]
[compile=*"true* | *false "*]
[factory=*"class_name"*]
[fileNS2pkg=*"file_name"*]
[genAll=*"true* | *false "*]
[genHelper=*"true* | *false"*]
[genImplTemplate=*"true* | *false"*]
[genRefrencedOnly=*"true* | *false"*]
[genSkeleton=*"true* | *false"*]
[genStub=*"true* | *false"*]
[gentestCase=*"true* | *false"*]
[gentypes=*"true* | *false"*]
[genWSDD=*"true* | *false"*]
[handlerFile=*"fileName"*]
[noImport=*"true* | *false "*]
[noWrapped=*"true* | *false"*]
[ns2pkg=*"package=namespace"*]
[outputDir=*"path"*]
[package=*"packageName"*]
[passwd=*"password"*]
[scope=*"Request* | *Application* | *Session"*]
[serverside=*"true* | *false"*]

[timeout=*"seconds"*]
[tm=*"argument"*]
[typeMappingVer=*"1.1* | *1.2"*]
[user=*"userName"*]
WSDLURI=*"resourceIdentifier"* >

Where:

Option	Description
classpath	Specify the classpath in quotes.
compile	If true, compiles the generated source code.
factory	Name of the class file that implements the GenerateFactory class.

fileNS2pkg	The name of the file that contains the ns2pkg (namespace to package) mappings. Use this option instead of the ns2pkg options to declare multiple mappings. For example, the *Ns2pkg.properties* file contains two mappings:
	`http\://Host:Port/Man.xsd=com.sybase.manf`
	`http\:/Host:Port/Purch.xsd=com.sybase.Purchase`
	and can be used as follows:
	wstool wsdl2java -fileNs2pkg Ns2pkg.properties myTest.wsdl

genWSDD	If true, generates a *Deploy.wsdd* file.
genImplTemplate	If true, generates a template for the implementation code.
genStub	If true, generates the stub files.
genAll	If true, generates and compiles the stubs, *wsdd*, and *ImplTemplate* files. If set to true, this option overrides the settings of `genWSDD`, `genImplTemplate`, and `genStub`. **Note** When user defined types that are not Java beans are used, the generated test client is not compilable as wsdl2java cannot construct the type in the test code.
gentestCase	If true, generates a test case.
gentypes	Set this option to false when you start with java2wsdl, or you will overwrite existing types. Default is true.
genHelper	If true, generates helper classes for metadata.
genSkeleton	If true, generates the skeleton files.
handlerFile	The handler class file that contains any special routines (handlers) for this Web service.
noImports	If true, generates code for the current WSDL only.

Option	Description
noWrapped	If true, turns off support for "wrapped" document/literal. Wrapped is a document literal variation, that wraps parameters as children of the root element.
ns2pkg	The namespace to package value pair, in the form *namespace=package*. You can only declare one namespace to package pair using this option. Use the fileNS2pkg option to declare multiple mappings.
outDir	The output directory for the generated files.
package	The package name to be used for namespace to package mappings.
passwd	The password required by the user to access the WSDL URI.
scope	The scope of the *deploy.wsdd*: request, application, or session.
serverside	If true, generates the server-side bindings for the Web service.
timeout	In seconds, the amount of time allowed for this command to complete before timing out.

tm	specify the type mapping file name, if any custom data types are being used. For example, the type mapping file *myTMfile.map* has the following contents:
	`t1.QName = nonbeansample:Book`
	`t1.Serializer = nonbeansample.BookSerializer`
	`t1.Deserializer = nonbeansample.BookDeserializer`
	`t1.SerializerFactory = nonbeansample.BookSerFactory`
	`t1.DeserializerFactory = nonbeansample.BookDeserFactory`
	`t1.TypeName = nonbeansample.Book`
	`t1.EncodingType = http://schemas.xmlsoap.org/soap/encoding/`
	`# Specify the webservice if the type`
	`# mappings are on the server`
	`t1.ServiceName = myCollection/myService`

typeMappingVer	Type mapping version to use. The default is 1.1. Acceptable values are 1.1 and 1.2.
user	The user name used to access the WSDL URI.

Examples

This example uses *CodeGetTest.wsdl* as the input WSDL file and generates the Java output file to the *out* directory:

```
wstool  wsdl2java -genTestCase false -genHelper true -genImplTemplate true
-genRefrencedOnly false -genSkeleton true -genStub true -genWSDD true -tm
tmfile.map -classpath "d:\out;d:\wstool\test\tm\classes" -genall false -
```

```
outDir out CodeGenTest.wsdl
```

Ant build example:

```
<wst_antTask command="wsdl2java"
entity="d:\wstool\test\sample.wsdl" />
```

java2Wsdl

Description

Generates code for client side artifacts from the Java class file, where locationURL and JavaClassName are the URL and name of the Java class file from which the WSDL is being generated.

Syntax

Command line:

java2wsdl
[-binding *binding_name*]
[-classpath *path*]
[-exposeMethods *m1, m2, m3*]
[-extraClass *class1, class2, class3*]
[-importURL *wsdl_interface*]
[-impINS *implementation_namespace*]
[-implWSDL *implementation_wsdl_filename*]
[-implClass *class_name*]
[-inheritMethods *true* | *false*]
[-inputSchema *file_or_url*]
[-inputWSDL *WSDL_file*]
[-intfNS *interface_namespace*]
[-outputWsdl *file_name*]
[-pkg2ns *package_namespace*]
[-portName *port_name*]
[-portTypeName *class_name*]
[-serviceName *service_name*]
[-soapAction *Default* | *Operation* | *None*]
[-stopClasses *class1, class2, class3*]
[-style *Document* | *RPC* | *Wrapped*]
[-tm *argument*]
[-typeMappingVer *1.1* | *1.2*]
[-use *Literal* | *Encoded*]
[-wsdlMode *All* | *Interface* | *Implemenation*]
[-xcludeMethods *m1, m2, m3*]
-locationURL<service location URL> javaClassName

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="java2wsdl" > <wst_antTask command="java2wsdl"

[binding=*"binding_name"*]
[classpath=*"path"*]
[exposeMethods=*"m1, m2, m3 "*]
[extraClass=*"class1, class2, class3"*]
[importURL=*"wsdl_interface"*]
[impINS=*"implementation_namespace"*]
[implWSDL=*"implementation_wsdl_filename"*]
[implClass=*"class_name*]
[inheritMethods=*"true | false"*]
[inputSchema=*"file_or_url"*]
[inputWSDL=*"WSDL_file"*]
[intfNS=*"interface_namespace"*]
[outputWsdl=*"file_name"*]
[pkg2ns=*"package_namespace"*]
[portName=*"port_name"*]
[porTypetName=*"class_name"*]
[serviceName=*"service_name"*]
[soapAction=*"Default | Operation | None"*]
[stopClasses=*"class1, class2, class3 "*]
[style=*"Document | RPC | Wrapped"*]
[tm=*"argument"*]
[typeMappingVer=*"1.1 | 1.2"*]
[use=*"Literal | Encoded"*]
[wsdlMode=*"All | Interface | Implemenation"*]
[xcludeMethods=*"m1, m2, m3"*]
locationURL<service location URL>=*"javaClassName" >*

Where:

Option	Description
binding	The binding name. The default is servicePortName value "SOAPBinding."
classpath	Specify the classpath in quotes.
exposeMethods	A comma-separated list of methods to expose.
extraClasses	A comma-separated list of classes to be added to the type section.
importURL	The location of the interface URL.
implNS	The target namespace for the implementation WSDL.
intfNS	The target namespace.
inputWSDL	input WSDL filename.
implWSDL	The output implementation WSDL file name. Setting this option causes the `wsdlMode` option to be ignored.
implClass	An optional class that contains implementation of methods in class-of-portType. The debug information in the class is used to obtain the method parameter names, which are used to set the WSDL part names.
inputWsdl	The input WSDL file name.

Option	Description
outputWsdl	The output WSDL file name.
pkg2NS	The package to namespace value pair, in the form *package=namespace*.
portName	The service port name. The default is obtained from the *locationURL*.
portTypeName	The port type name. The default is class-of-portType.
serviceName	The service name. The default is servicePortName value “Service.”
inheritMethods	True or false. If true, expose allowed methods in inherited classes.
xcludeMethods	A comma-separated list of methods not to expose.
stopClasses	A comma-separated list of class names that stops the inheritance search even if the `inheritMethods` option is specified.

tm	specify the Type mapping file name, if any custom data types are being exposed. For example, the type mapping file *myTMfile.map* has the following contents:
	`t1.QName = nonbeansample:Book`
	`t1.Serializer = nonbeansample.BookSerializer`
	`t1.Deserializer = nonbeansample.BookDeserializer`
	`t1.SerializerFactory = nonbeansample.BookSerFactory`
	`t1.DeserializerFactory = nonbeansample.BookDeserFactory`
	`t1.TypeName = nonbeansample.Book`
	`t1.EncodingType = http://schemas.xmlsoap.org/soap/encoding/`
	`# Specify the webservice if the type`
	`# mappings are on the server t1.ServiceName = myCollection/myService`

typeMappingVer	The type mapping version. Valid options are 1.1 (the default) and 1.2.
soapAction	The value of the operations soapAction field. Valid values are: Default – causes the soapAction to be set according to operations in the metadata. Operation – forces soapAction to the name of the operation. None – forces the soapAction to blank, which is the default.
style	The style of the binding in the WSDL. Options are “Document,” “Wrapped,” or “RPC” (the default).
use	Defines the use of the items in the binding. Options are “Literal” or “Encoded” (the default).

Option	Description
wsdlMode	The output WSDL mode. Valid options are All (default), Interface, or Implementation.
inputSchema	A file or URL that points to the XML schema used during WSDL generation.

Examples

This example uses *nonBeanSample* as input and generates the *CodeGenTest.wsdl* output file:

```
wstool java2wsdl  -locationURL
"http://localhost:8080/nonBean/services/nonBeanSample"  -pkg2ns
"nonbeansample=nonbeansample" -tm tmfile.map -outputwsdl CodeGenTest.wsdl
-classpath d:\wstool\test\tm\classes nonbeansample.TestBookServiceIntf
```

Ant build example:

```
<wst_antTask command="java2wsdl"
locationURL="http://${wst.host}/${wst.port}/nonBean/se
rvices/nonBeanSample"
tm="d:\wstool\test\tm\tmfile.map"
classpath="d:\wstool\test\classes"
entity="nonbeansample.TestBookServiceIntf"/>
```

Security commands

Description

Security administration commands allow you to perform security-related Web service management.

Command list

Table 8-5 lists the security administration commands.

Table 8-5: wstool security administration commands

command name	Description
add	Adds existing roles to a method or Web service.
remove	Removes a role from a method or Web service.

Note To run security administration commands, you must be a member of the Admin Role.

add

Description

Adds existing roles to a Web service or Web service method.

Syntax

Command line:

```
add
[-sourceEntity r1, r2, r3]
TargetEntity
```

Ant build file:

```
<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="add" > <wst_antTask command="add"
[sourceEntity="r1, r2, r3"]
TargetEntity="entity" >
```

Where:

Option	Description
sourceEntity	A comma-separated list of roles to be added in the format of: role:*role1*, *role2*
targetEntity	The target entity to which the roles are assigned: • method:*CollectionName/ServiceName/MethodName* – identifies the method to which the roles are added. • service:*CollectionName/ServiceName* – identifies the Web service to which the roles are added.

Examples

Example 1 This example adds role1 and role2 to myMethod:

```
wstool add "role:role1, role2" "method:WebColl/WebServ/myMethod"
```

Example 2 This example adds the role everybody to myWebServ:

```
wstool add "role:everybody" "service:WebColl/myWebServ"
```

Ant build example:

```
<wst_antTask command="add"
sourceentity="role:testRole"
targetentity="service:myCollection/myService"/>
```

Note Entities must be specified in quotes.

remove

Description

Removes roles from a Web service or Web service method.

Syntax

Command line:

remove
[-sourceEntity *r1, r2, r3*]
TargetEntity

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="remove" > <wst_antTask command="remove"
[sourceEntity=*"r1, r2, r3"*]
TargetEntity="*entity*" >

Where:

Option	Description
sourceEntity	A comma-separated list of roles to be removed from the format of: role:*role1*, *role2*
targetEntity	The target entity to which the roles are removed: • method:*CollectionName/ServiceName/MethodName* – identifies the method from which the roles are removed. • service:*CollectionName/ServiceName* – identifies the Web service from which the roles are removed.

Examples

Example 1 This example removes role1 and role2 from myMethod:

```
wstool remove "role:role1, role2" "method:WebColl/WebServ/myMethod"
```

Example 2 This example removes the role everybody from myWebServ:

```
wstool remove "role:everybody" "service:WebColl/myWebServ"
```

Ant build example:

```
<wst_antTask command="remove"
sourceentity="role:testRole"
targetentity="service:myCollection/myService"/>
```

Note Entities must be specified in quotes.

wstkeytool commands

Description

This section contains information on wstkeytool commands, and lists the commands that wstkeytool accepts. Each command has a brief description, a list of options, and an example of its usage.

Each command has its own section heading (the text in the far left margin). 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. Table 8-6 lists the wstkeytool commands.

Table 8-6: wstkeytool command

Command	Description
changePin	Changes the password of the keystore.
deleteCert	Deletes a key from the keystore.
export	Exports a certificate from the keystore to a file.
genCertReq	Generates a certificate request using PKCS#10 format.
GetCACerts	Lists the CA certificates contained in the keystore.
GetOtherCerts	Lists the Other certificates contained in the keystore.
GetUserCerts	Lists the User certificates contained in the keystore.
import	Imports a certificate from a file to the keystore.
printCert	Prints the certificate information from a file.

Note To run wstkeytool commands, you must belong to the Admin Role.

changePin

Description

Changes the password used to protect the integrity of the keystore contents.

Syntax

Command line:

```
changePin
[-newpin new_pin]
```

Ant build file:

```
<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
```

<target name="changePin" > <wst_antTask command="changePin" [newpin=*"new_pin"* >]

Where:

Option	Description
`newpin`	The new keystore password

Examples This command changes the keystore password to "new_password:"

```
wstkeytool changePin -newpin new_password
```

Ant build example:

```
<wst_antTask command="changepin" newpin="secret" />
```

deleteCert

Description Deletes a certificate identified by the entry identified by *keyname* from the keystore.

Syntax **Command line:**

deleteCert
keyname

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="deleteCert" > <wst_antTask command="deleteCert"
keyname="*key*" >

Examples This command deletes the certificate identified by testkey from the keystore:

```
wstkeytool deleteCert testkey
```

export

Description Reads the certificate associated with the entity stored in the keystore and exports it to a file.

Syntax **Command line:**

```
export
[-p12 pin]
[-certFile file_name]
[-encoding encodingFormat]
[-format exportFormat]
[-includeCertChain true | false]
[-privPbeAlgo algorithm]
[-certPbeAlgo algorithm]
entity
```

Ant build file:

```
<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="export" > <wst_antTask command="export"
[p12="pin" ]
[certFile="encodingFormat"]
[encoding="encodingFormat"]
[format="exportFormat"]
[includeCertChain="true | false" ]
[privPbeAlgo="algorithm" ]
[certPbeAlgo="algorithm"]
entity="entity" >
```

Where:

Option	Description
`p12`	This option must be used if the type of certificate is pkcs12. Valid arguments are: • PBEWithSHA1And3KeyTripleDESCBC • PBEWithSHA1And2KeyTripleDESCBC • PBEWithSHA1And128BitRC2CBC • PBEWithSHA1And128BitRC4 • PBEWithSHA1And40BitRC4
`certFile`	File name with extension of the exported certificate generated in the current directory.
`format`	The format of the exported certificate. Options are: • der (default) • netxcape • pkcs7 • pkcs12
`encoding`	The encoding of the exported certificate. Either binary (default), or base64.
`includeCertChain`	True or false. If true, includes certificate chain information. The default is true.

Option	Description
`privPbeAlgo`	The private pbe's algorithm. Possible values include: • PBEWithSHA1And3KeyTripleDESCBC • PBEWithSHA1And2KeyTripleDESCBC • PBEWithSHA1And128BitRC2CBC • PBEWithSHA1And40BitRC2CBC • PBEWithSHA1And128BitRC4 • PBEWithSHA1And40BitRC4
`certPbeAlgo`	The certificate pbe's algorithm. The possible values are the same as `privPbeAlgo`.
Entity	The exported certificate in the form key:*certificate*

Examples

This example exports information from the certificate *mytest.crt* from the keystore, and exports it to the file *exported_cert*:

```
wstkeytool export -format pkcs7 -certFile exported_cert -p12 Pin
-privPbeAlgo PBEWithSHA11And3keyTripleDESCBC -certPbeAlgo
PBEWithSHA11And3keyTripleDESCBC -certfile d:\mykeys\mytest.crt key:"Sybase
Jaguar User Test Certificate"
```

genCertReq

Description

Generates a Certificate Signing Request (CSR), using the PKCS#10 format. The key is stored as *keyname* in the keystore, and stores the certificate request to the *certFile* if supplied.

Syntax

Command line:

genCertReq
[-certFile *file_name*]
[-emailId *email_address*]
[-isSensitive *true* | *false*]
[-phone *number*]
[-requestorName *name*]
[-serverAdmin *name*]
[-sigalg *sigalg_name*]
[-keysize *number*]
[-dname *distinguished_name*]
[-userId *name*]
keyname

Ant build file:

```
<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="genCertReq" > <wst_antTask command="genCertReq"
[certFile="file_name"]
[emailId="email_addresse" ]
[isSensitive="true | false"]
[phone="number" ]
[requestorName="name"]
[serverAdmin="name" ]
[sigalg="sigalg_name"]
[keysize="number"]
[dname="distinguished_name" ]
[userId="name"]
keyname="key" >
```

Where:

Option	Description
`sigalg`	The signature algorithm that defines the key algorithm used and the hash method used to compute the message digest. MD5withRSA is the default. If key is of type "DSA", the default is "SHA1withDSA", if key is of type "RSA", the default is "MD5withRSA"
`keysize`	The size indicates the authentication key strength. The greater the number, the stronger the encryption. Your options are 512, 768, or 1024. 1024 is the default.
`dname`	The distinguished name in the format: "CN=cName, OU=orgUnit, O=org, L=city, S=state, C=countryCode"
`userId`	An optional user name or ID.
`emailId`	An optional e-mail address.
`requestorName`	An optional requestor name.
`serverAdmin`	A optional server administrator name.
`phone`	An optional phone number.
`certFile`	An optional file name. If specified, the output is written to this file; if not, it is displayed on the console.

Examples

This example generates a CSR named certreq, using the signature algorithm md5withRSA. Information contained in the testcert key is used to generate the CSR.

```
wstkeytool genCertReq -sigalg md5withRSA keysize 1024 -certFile testcert
testcert
```

GetCACerts

Description — Lists the CA (certification authority) certificates contained in the server.

Syntax — **Command line:**

GetCACerts

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="GetCACerts" > <wst_antTask command="GetCACerts" >

Examples — This command lists the CA certificates contained in the server:

```
wstkeytool GetCACerts
```

Ant build example:

```
<wst_antTask command="getcacerts" />
```

GetOtherCerts

Description — Lists the other certificates (listed in the other certificates folder) contained in the server.

Syntax — **Command line:**

GetOtherCerts

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="GetOtherCerts" > <wst_antTask
command="GetOtherCerts" >

Examples — This command lists the other certificates contained in the server:

```
wstkeytool GetOtherCerts
```

Ant build example:

```
<wst_antTask command="getOthercerts" />
```

GetUserCerts

Description — Lists the user certificates (listed in the User folder) contained in the server.

Syntax — **Command line:**

GetUserCerts

Ant build file:

```
<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="GetUserCerts" > <wst_antTask
command="GetUserCerts" >
```

Examples — This command lists the user certificates contained in the server:

```
wstkeytool GetUserCerts
```

Ant build example:

```
<wst_antTask command="getusercerts" />
```

import

Description — Reads the certificate from a file and stores it as a keystore entry.

Syntax — **Command line:**

import
[-p12 *pin*]
filename

Ant build file:

```
<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="import" > <wst_antTask command="import"
p12= "pin"
filename="file" >
```

Where:

Option	Description
`p12`	This option must be used if the imported certificate is pkcs12 (the certificate file ends with *.p12* or *.pfx*).

Option	Description
`filename`	The file that contains the information being imported into the keystore. File type can be base64, binary, der, pkcs7, or Netscape.

Examples

This example imports a certificate from *import_cert.crt*, and stores it in the keystore:

```
wstkeytool import imported_cert.crt
```

Ant build example:

```
<wst_antTask command="importcert"
entity="d:\test\test.crt"/>
```

printCert

Description

Reads the certificate information from the keystore, and prints its contents in a human-readable format to the display. keyname is the name of the printed certificate.

Syntax

Command line:

printCert
keyname

Ant build file:

<taskdef name="wst_antTask"
classname="com.sybase.wst.wstool.ant.AntTask"/>
<target name="printCert" > <wst_antTask command="printCert"
keyname=*"key"* >

Examples

This command displays the information contained in the Jaguar user test CA:

```
wstkeytool printCert "Sybase Jagauar User Test CA"
```

Ant build example:

```
<wst_antTask command="printcert" entity="Sybase Jaguar
User Test CA"/>
```

CHAPTER 9

Developing Web Service Clients

This chapter describes how to develop Web service clients from the client files created from the WST development tool and wstool commands.

Introduction

When you use Web Services Toolkit to generate client files, you generate a variety of files based on the options selected and the client model used. This chapter describes how to create Web service client applications based on various programming models, including:

- “Stub-based model client” on page 154
- “Dynamic proxy client” on page 156
- “Dynamic invocation interface client” on page 157
- “Document style client” on page 161

Stub-based model client

The stub-based model generates local stub classes for the proxy from a WSDL document. This is the model used by the WST development tool to create a Web service client. When you change the WSDL document, you must regenerate the stubs. WST provides tools to generate and compile stubs. See “Creating and managing Web service clients” on page 38 and wsdl2Java on page 134. Along with the stubs, the tools generate additional classes, and a service definition interface (SDI), which is the interface that is derived from a WSDL’s portType. This is the interface you use to access the operations on the Web service. The combination of these files are called client-side artifacts. Client-side artifacts are a collection of files on the client-side that handle communication between a client and a Web service.

Generated client-side artifacts must include:

- A stub class – for example, *AddNumbersStub.java*:

```
public class AddNumbersStub extends org.apache.axis.client.Stub
implements client.AddNumbers_Port
```

- A service endpoint interface – for example, *AddNumbers_Port.java*:

```
public interface AddNumbers_Port extends java.rmi.Remote
```

- A service definition interface – for example, *AddNumbers_Service.java*:

```
public interface AddNumbers_Service extends javax.xml.rpc.Service
```

- An implementation of the service definition interface (the location class to help you find the endpoint) – for example, *AddNumbers_ServiceLocator.java*:

```
public class AddNumbers_ServiceLocator extends
org.apache.axis.client.Service implements client.AddNumbers_Service
```

stub-based example

This stub-based client example can be found in *%JAGUAR%\sample\wst_samples\JavaClassClient\client\AddClient.java*.

```
package client;


import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
```

```
/**
* @author Sybase
*
* A sample client to access the add method in the service.
* The add method returns the addition of two numbers
* which is calculated by the webservice AddNumbers.
*
*/
public class AddClient
{
    public static void main(String[] args)
    {
        AddNumbers_ServiceLocator context;
        AddNumbers_Port client;
        try
        {
            int num1 = 10;
            int num2 = 10;
            context = new AddNumbers_ServiceLocator();
            client = context.getAddNumbers();

            if (args.length > 0)
            {
                num1 = new Integer(args[0]).intValue();
                if (args.length >1)
                {
                    num2 = new Integer(args[1]).intValue();
                }
            }
            int value= client.add(num1, num2);
           System.out.println("Result of adding " + num1 + " and " +  num2  +
" is: " + value);
        }
        catch (ServiceException ex1)
        {
            ex1.printStackTrace();
        }
        catch (RemoteException ex2)
        {
            ex2.printStackTrace();
        }
    }
}
```

Dynamic proxy client

The dynamic proxy client creates dynamic proxy stubs at runtime using JAX-RPC client APIs. The client gets the service information from a given WSDL document. It uses the service factory class to create the service based on the WSDL document and obtains the proxy from the service.

The significant JAX-RPC client APIs used are:

- javax.xml.rpc.rpc.Service
- javax.xml.rpc.ServiceFactory

Dynamic proxy client example

This section contains a listing of a sample dynamic proxy client:

```
package client;

import java.net.URL;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;


public class JAXRPC_DynClient
{
    public static void main(String[] args)
    {
        try
        {
       /*
        * URL to the Service's WSDL document
       */
            URL wsdl = new
URL("http://localhost:8080/ws/services/AddNumbers?wsdl");
            String namespaceURI = "urn:simpleJavaClass.AddNumbers";
            String serviceName = "AddNumbers";
            String portName = "AddNumbers";
            int num1 = 10;
            int num2 = 20;


            ServiceFactory factory = ServiceFactory.newInstance();

            /*
```

```
* Create a service using the WSDL document
*/
          Service service = factory.createService(wsdl, new QName(namespaceURI,
serviceName));

       /*
        *  Get the proxy by calling getPort method from an instance of Service.
       */
            AddNumbers_Port port = (AddNumbers_Port)service.getPort(new
QName(namespaceURI, portName), AddNumbers_Port.class);
            int value= port.add(num1, num2);
           System.out.println("Result of adding " + num1 + " and " +  num2  +
" is: " + value);

        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

}
```

Dynamic invocation interface client

The Dynamic Invocation Interface (DII) client does not require a WSDL file to generate static stubs or pass the WSDL file to the service factory to create the service; instead, the client must know a service's address, operations, and parameters in advance. A DII client discovers service information dynamically at runtime by a given set of service operations and parameters as you will see in the example.

The significant JAX-RPC client APIs used are:

- javax.xml.rpc.Call
- javax.xml.rpc.Service
- javax.xml.rpc.ServiceFactory

DII client example

This section contains a listing of several sample DII clients.

In addition, you can find a DII sample in *%JAGUAR%\sample\wst_samples\JAXRPCClient\client\DIIClient.java*

```
package client;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.ParameterMode;
import javax.xml.namespace.QName;

public class JAXRPC_DIIClient
{
    public static void main(String[] args)
    {
        try
        {
            /*
       *  URL of the web service
       */
            String address = "http://localhost:8080/ws/services/AddNumbers";
            String namespaceURI = "urn:simpleJavaClass.AddNumbers";
            String serviceName = "AddNumbers";
            String portName = "AddNumbers";
            int num1 = 10;
            int num2 = 20;


            ServiceFactory factory = ServiceFactory.newInstance();

       /*
       * Create an instance of the Service with the given service QName
       */
            Service service = factory.createService(new QName(serviceName));

            Call call = service.createCall(new QName(portName));

            call.setTargetEndpointAddress(address);

         QName intQName = new QName("http://www.w3.org/2001/XMLSchema", "int");
            /*
        * Set operation name to invoke.
        */
            call.setOperationName(new QName(namespaceURI,"add"));
```

```
            /*
        * Add parameters definitions in the call object.
        */
            call.addParameter("number1", intQName, ParameterMode.IN);
            call.addParameter("number2", intQName, ParameterMode.IN);

            /*
        * Set definition of the return type.
        */
            call.setReturnType(intQName);

            Object[] inParams = new Object[2];
            inParams[0] = new Integer(num1);
            inParams[1] = new Integer(num2);

            int value= ((Integer)call.invoke(inParams)).intValue();
           System.out.println("Result of adding " + num1 + " and " +  num2  +
" is: " + value);

        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

}
```

Significant AXIS client APIs used for the following sample are:

- org.apache.axis.client.Call
- org.apache.axis.client.Service

The sample client is in *%JAGUAR%\sample\wst_samples\DynamicClient\client\DynClient.java*

```
package client;
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import javax.xml.rpc.ParameterMode;
import java.net.URL;

/**
 * @author Sybase
 */
public class DynClient
```

```
{
    public static void main(String[] args) throws Exception
    {
        /*
        * URL of the web service.
        * See under eclipse ->Web services perspective ->'ws'->AddNumbers
        */
        String url = "http://localhost:8080/ws/services/AddNumbers";
        Integer number1 = new Integer(1);
        Integer number2 = new Integer(2);


        /*
        * Create an instance of the Call object.
        */
        Call call = (Call)service.createCall();
        System.out.println("Connecting to: " + url);
        call.setTargetEndpointAddress(new URL(url));
        /*
        * Set parameters definitions in the call object.
        */
        call.addParameter("n1", XMLType.XSD_INT, ParameterMode.IN);
        call.addParameter("n2", XMLType.XSD_INT, ParameterMode.IN);
        /*
        * Set definition of the return type.
        */
        call.setReturnType(XMLType.XSD_INT);
        /*
        * Name of the method to invoke.
        */
        call.setOperation("add");
        System.out.println("Adding: " + number1 + " & " + number2);
        /*
        * Finally invoke the method.
        */
        Integer i = (Integer)call.invoke(new Object[]
        {
            number1, number2
        }
        );
        System.out.println("Result: "+ i);
    }
}
```

Document style client

The previous client examples use different invocation modes to interact with RPC style Web services. To interact with document style Web services, the XML document must be defined in the client. The clients do not invoke the Web service by sending a discrete set of parameters and receiving return values as described in a WSDL document; instead, they send the parameter to the service as XML documents.

Document style example

This section contains document style client examples.

```
package client;
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import javax.xml.namespace.QName;
import javax.xml.rpc.ParameterMode;
import org.apache.axis.enum.Style;
import org.apache.axis.enum.Use;
import org.apache.axis.message.SOAPBodyElement;
import org.apache.axis.message.SOAPEnvelope;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.apache.axis.utils.XMLUtils;
import java.util.Vector;

public class DIIDocClient
{

    public static void main(String[] args)
    {
        try
        {
            /*
       * NOTE: The web service uses document style
       * eg:
       *<service name="MyDocSample" provider="java:RPC" style="document"
use="literal">
       */
            String url = "http://localhost:8080/ws/services/MyDocSample";
            Service service = new Service();
            Call call = (Call) service.createCall();
```

```
            call.setTargetEndpointAddress(url);
            String param= "hello";

       /*
        *construct the XML document
       */
            SOAPBodyElement[] input = new SOAPBodyElement[1];

            input[0] = new
SOAPBodyElement(XMLUtils.StringToElement("http://www.w3.org/2001/XMLSchema",
                "echo", param));
            Vector          elems = (Vector) call.invoke( input );

            SOAPBodyElement elem  = (SOAPBodyElement) elems.get(0);
            Element e = elem.getAsDOM();
          System.out.println("returned value: " + XMLUtils.ElementToString(e));

        }
        catch (Throwable t)
        {
            t.printStackTrace();
        }
    }
}
```

In the above example, the XML in the request sent to the server is:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
       <soapenv:Body>
<xsd:echo>hello</xsd:echo>
</soapenv:Body>
</soapenv:Envelope>
```

If the example is RPC-style, the XML in the request sent to the server is:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
       <soapenv:Body>
          <echo xmlns="">
             <arg0 xsi:type="xsd:string">hello</arg0>
```

```
          </echo>
       </soapenv:Body>
</soapenv:Envelope>
```

CHAPTER 10

Using the Web Services Toolkit Samples

This chapter describes the samples and tutorials included with WST.

Samples in WST

WST includes sample Web services that guide you through using the WST development tool, deploying Web services, establishing security constraints, creating clients, and so on.

Samples on the Sybase Web site

You can find additional EAServer samples on the EAServer CodeXchange pages at http://easerver.codexchange.sybase.com/. The CodeXchange site allows Sybase users to share code samples and utilities for EAServer and other Sybase products.

Sample and tutorial location

WST is included as part of an EAServer installation, if you select the Web Services option. Samples and tutorials are installed in:

- Windows – *%JAGUAR%\sample\wst_samples*
- UNIX – *$JAGUAR/sample/wst_samples*

Each sample and tutorial contains its own subdirectory that contains a *Readme.txt* file that describes the sample and how to run it, and the files necessary to run the sample.

Creating the sample projects and installing the samples

Before running any of the samples described in "Using the WST development tool and features" on page 166, you need to create the projects and install the samples in the WST development tool environment. From the WST development tool:

1 Select Sybase Web Services and click OK.

2 Select Help | Welcome.

3 Scroll down to "Sample Projects" and click the "here" link.

The projects are created and populated with samples which you can see from the Package Explorer (Window | Show View | Package Explorer).

Using the WST development tool and features

Your WST installation includes the following samples and tutorials that familiarize you with the WST development tool. All samples in this section require Eclipse to be running and the Sybase Web services plug-in installed. See Chapter 4, "Web Services Administration."

Exposing a Java class as a Web service

The WST development tool provides the architecture and tools to create and deploy Web services from various types of files, which generate a *WSDL* file as part of the Web service creation. From the generated *WSDL* file, use the WST development tool to create a test client that tests your newly exposed Web service.

In this tutorial, you:

- Connect to EAServer (Web services container)
- Expose a Java class as a Web service

- Generate a test client for the exposed Web service
- Run the test client

The Java source files are in the *JavaClass/simpleJavaClass* and *JavaClassClient* subdirectories. Instructions for running the tutorial are in the file *ReadmeForEclipse.txt* in the *JavaClass/simpleJavaClass* directory.

Running the test client using HTTPS

The sample described above includes a client, *AddClient.java* located in the *JavaClassClient/client* subdirectory. This section describes how to modify this sample and import a test certificate into a keystore so that you can run the tutorial using HTTPS.

To run the test client using HTTPS, you must have Java Secure Socket Extension (JSSE) installed and configured on the client. See Chapter 5, "Using SSL in Java Clients", in the *EAServer Security Administration and Programming Guide* for more information.

❖ Exporting the Jaguar Test CA

You must export the Jaguar Test CA using Security Manager. See the *EAServer Security Administration and Programming Guide* for information about starting and using Security Manager.

1 From Security Manager, select the CA Certificates folder.

2 Highlight the Sybase Jaguar User Test CA.

3 Select File | Export Certificate.

4 From the Export Certificate wizard, select the format type for the exported certificate. For the Test CA, select Binary Encode X509 Certificate. Click Next.

5 Select Save to File and enter the full path name to a file that will contain the test CA. Use *EASTestCA* as the certificate name.

Do not add any extension to the file name. A .crt extension is automatically added to the exported certificate by Security Manager.

6 Click Finish to export the certificate to the *EASTestCA.crt*.

❖ Create a Java Keystore containing the Sybase Jaguar test CA and mark the certificate trusted.

This procedure uses the Java keytool command to create a keystore, import the *EASTestCA.crt* certificate, and mark it trusted.

1 From the command line, go to the *$JAGUAR/sample/wst_samples* directory.

2 Enter this command to create the keystore named *EASTestCA.jks* and install *EASTestCA.crt*, mark it trusted, and protect the keystore with the password “changeit”:

```
<path_to_JDK_1.3>/bin/keytool -import -v -trustcacerts -alias eastestca -file
EASTestCA.crt -keypass changeit -keystore EASTestCA.jks
```

The Java keytool command requires you to answer two questions. Here are the questions, answers you should provide, and output:

```
Enter keystore password: changeit
Owner: L=Sybase Jaguar User Test Locality, O=Sybase Jaguar User Test,
CN=Sybase Jaguar User Test CA (TEST USE ONLY)
Issuer: L=Sybase Jaguar User Test Locality, O=Sybase Jaguar User Test,
CN=Sybase Jaguar User Test CA (TEST USE ONLY)
Serial number: 1
Valid from: Fri Oct 16 11:02:16 PDT 1998 until: Thu Oct 16 11:02:16 PDT
2003
Certificate fingerprints:
MD5: 5B:66:65:6A:4F:11:41:7C:B4:9B:17:CF:30:61:26:5F
SHA1: B5:38:55:36:E2:FF:F2:28:5E:45:80:94:BF:54:20:96:28:5B:CC:F8
Trust this certificate? [no]: yes
Certificate was added to keystore
[Saving EASTestCA.jks]
```

❖ **Modify the client program**

Make these changes to the *AddClient.java* file. When you run the program, the client will connect to the listener at port 8081. This tutorial assumes that you are running the client on the same machine as your EAServer installation.

- modify *AddClient.java* as follows:

```
import java.net.URL;
import java.net.MalformedURLException;
import java.security.Security;
import java.io.File;


public class AddClient
{
public static void main(String[] args)
{

...
String jksStore=
```

```
".." + File.separator + ".." + File.separator +
"EASTestCA.jks";

System.setProperty("javax.net.ssl.trustStore", jksStore);
System.out.println("Set system property " +
javax.net.ssl.trustStore to " + jksStore);
// Dynamically register the JSSE provider
Security.addProvider(new
com.sun.net.ssl.internal.ssl.Provider());

context = new AddNumbers_ServiceLocator();
URL newURL = null;
try
{
newURL =
new URL("https://localhost:8081/AddSample/services/AddNumbers");
}
catch(MalformedURLException me)
{
me.printStackTrace();
return;
}

System.out.println("Connecting to: " + newURL.toString());
client = context.getAddNumbers(newURL);
```

❖ **Compile and run the sample**

- Change to the *JavaClassClient/client* subdirectory and compile and run the test client. For example:

```
$JAGUAR/bin/wstant compile run -Dnum1=5 -Dnum2=8
```

Exposing a Web service that implements JAX-RPC defined interfaces

The WST development tool allows you to develop, deploy, and run Web services that conform to the JAX-RPC specification.

In this tutorial, you:

- Deploy the Web service described in “Establishing Web service security, and generating a test client” on page 170
- Deploy a Java class as a Web service

- Run the JAX-RPC client

The source files are in the *JAXRPCService* and *JAXRPCClient* subdirectories. Instructions for running the tutorial are in the file *readmeEclipse.txt* in the *JAXRPCService/server* directory.

Exposing a stateless EJB as a Web service

The WST development tool offers different options for exposing Web services and generating test clients.

In this tutorial, you:

- Use EAServer Manager to install an EJB into EAServer
- Use the Quick Expose option to expose your EJB as a Web service
- Generate a JSP test client for the exposed Web service
- Run the JSP test client

The source files are in the *EJBSample/statelessEjbSample* subdirectory. Instructions for running the tutorial are in the file *readme.txt*.

Establishing Web service security, and generating a test client

Authorization for Web services is enforced by roles. Using the WST development tool you can create new roles, or use predefined roles to establish security at the Web service and Web service operation level. Using different levels of security at the operation level allows you to closely control who has access to your resources.

In this tutorial, you:

- Expose a Java class as a Web service
- Create an administrator role and a general role and assign those roles to Web service operations/methods
- Generate two test clients: one to administer, the other for general business purposes
- Run the test clients to verify that the security constraints provided by the roles work as expected

The source files are in the *SecuritySample* and *SecuritySampleClient* subdirectories. Instructions for running the tutorial are in the file *readme.txt* in the *SecuritySample* directory.

Exposing a CORBA component as a Web service

This sample demonstrates how to expose an existing CORBA component as a Web service. In addition, the sample illustrates which component properties enable the component to be exposed. The WST development tool contains an "Other components" folder, which contains components that can be deployed as Web services. Once deployed, the components remain listed in the Other Components folder. This tutorial contains a CORBA Java component that can be exposed as a Web service once it is made stateless. Use EAServer Manager and the WST development tool together to run this tutorial.

In this tutorial, you:

- Use EAServer Manager to install a CORBA component in EAServer
- Modify the CORBA component's properties so it is stateless
- Expose the component as a Web service
- Create and run a Web service client

The source files are in the *CORBAComponent* subdirectory. Instructions for running the tutorial are in the file *readme.txt*.

Developing client applications

This section provides samples of client applications that you can use to access the Web services contained on EAServer.

In addition to the samples described in this chapter, see Chapter 9, "Developing Web Service Clients" for a description of various types of clients, including sample code.

Running a dynamic client

This sample is an example of using a dynamic client (a client that needs no client-side artifacts because it already knows the name and how to access the Web service method) to invoke your Web service's methods.

In this tutorial, you:

- Deploy and expose a Web service
- Run the dynamic client

The source files are in the *DynamicClient* subdirectory. Instructions for running the tutorial are in the file *readme.txt*.

.NET sample

Client applications created with Microsoft's .NET framework can access Web services hosted in EAServer; *wsdl.exe* generates the required client side proxy from the WSDL document on EAServer. The C# ("C-sharp") compiler *csc.exe* compiles the client executable program that accesses the Web service through the client-side proxy.

The example in this section describes the steps for creating both the server-side Web service and client-side proxy and executable program. For more information about .NET, go to Microsoft's .NET Web site at http://www.microsoft.com/net/.

Note Only primitive data types are supported as Web services accessible by .NET clients, not user-defined data types. See Chapter 3, "Components, Datatypes, and Type Mappings," for a list of supported data types.

In this sample, you:

- Deploy a Web service using an example from a previous tutorial for this step. See "Exposing a stateless EJB as a Web service" on page 170.
- Install Microsoft's .NET SDK on your machine and add the .NET libraries to your environment.
- Run .NET's wsdl.exe command or a batch file to create the client proxy.
- Create the client executable program and compile it using *csc.exe*.
- Run the client program to invoke the service.

The source files are in the *DotNetSample* subdirectory. Instructions for running the tutorial are in the file *readme.txt*.

APPENDIX A

Migrating 4.x Web Services

This appendix describes how to migrate Web services created with Web Services Toolkit version 4.*x* to be compatible with Web Services Toolkit version 5.0 and above.

Introduction

When you upgrade an existing EAServer 4.*x* installation to EAServer version 5.0 or later, any 4.*x* EAServer components and packages are automatically migrated to the 5.*x* EAServer.

To use your Web services in the 5.*x* server you must modify both the client application and the server-side Web service.

This appendix uses the CanineShelter example to illustrate how to migrate Web services and Web service client applications. The CanineShelter example is located in the */sample/wst_samples/CanineShelter* subdirectory of your EAServer installation.

All paths listed below are for Microsoft Windows. UNIX users should make the necessary adjustments.

Server-side migration

Migrating Web services on the server side requires you to:

1 Verify that the skeletons are generated for all of the migrated 4.*x* components. If any of the skeletons are not available, use jagtool or EAServer Manager to generate them. See “Generating Stubs and Skeletons” in the *EAServer Programmer’s Guide*.

2 Run the wstool upgrade command. This ensures that all the 4.*x* Web services are migrated to 5.0 Web services. See upgrade on page 134.

Let’s use the CanineShelter example to show how to perform the server-side migration:

1 Once you have installed EAServer 5.*x*, the SoapDemo package is migrated from 4.*x* to 5.*x*.

2 Verify the skeletons for the SoapDemo package are available, or generate the skeletons for the component using:

```
jagtool gen_skels Package:SoapDemo
```

3 Verify that the WSDL files, *SoapSample.wsdl* and *SoapSampleImpl.wsdl* are available in the directory *%JAGUAR%\Webservices\work\wsdl*.

4 Run the wstool upgrade command to upgrade the Web services:

```
wstool upgrade
```

5 List the collections on the server:

```
wstool list collections
```

You should see “SoapSample” listed.

This completes the server-side migration.

Client-side migration

If an existing 4.*x* client uses the Sybase soapproxy, you need to re-write the client. See Chapter 9, “Developing Web Service Clients.”

To migrate the CanineShelter client-side code, you must:

1 Rewrite the client-side code. See Chapter 9, “Developing Web Service Clients.” When you are done, the new client should resemble the sample client that is located in the *%JAGUAR%\samples\wst_samples\CanineShelter\client\src\com\Sybase\webservice\sample\soap\client* directory.

2 Run the sample. Follow the instructions located in the *Readme.html* file located in the *%JAGUAR%\samples\wst_samples\CanineShelter* directory.

Index

Symbols

A

B

C

D

E

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X