SYBASE®

Web Services Toolkit User's Guide

EAServer

6.0

DOCUMENT ID: DC31727-01-0600-01

LAST REVISED: July 2006

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 and Datatypes” – 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, “Management Console—Web Services” – the procedures for managing Web services from the Sybase Management console.
- Chapter 6, “Management console—Registry Services” – the procedures for managing UDDI registries from the Sybase Management console.
- Chapter 7, “Developing Web Service Clients” – description of various client application styles.
- Chapter 8, “J2EE Web Service Support” – the procedures for managing J2EE Web services.
- Chapter 9, “Using wstool and wstant” – description of how to use wstool command line tools.

Related documents

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

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

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

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

- Define and configure entities, such as EJB modules, Web applications, data sources, and servers
- Perform administrative and deployment tasks

The *EAServer CORBA Components Guide* explains how to:

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

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

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

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

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

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

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

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

- Understand the EAServer security architecture
- Configure role-based security for components and Web applications
- Configure SSL certificate-based security for client connections

- Implement custom security services for authentication, authorization, and role membership evaluation
- Implement secure HTTP and IIOP client applications
- Deploy client applications that connect through Internet proxies and firewalls

The *EAServer System Administration Guide* explains how to:

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

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

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

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

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

Conventions

The formatting conventions used in this manual are:

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

Other sources of information

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

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

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

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

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

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

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

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

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

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

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

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

Sybase EBFs and software maintenance

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

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

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

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

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

Accessibility features

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

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

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

1 Start Eclipse.

2 Select Help | Help Contents.

3 Enter `Accessibility` in the Search dialog box.

4 Select Accessible User Interfaces or Accessibility Features for Eclipse.

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

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

If you need help

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

CHAPTER 1

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.

SAAJ 1.1

The SOAP with Attachments API for Java 1.1 (SAAJ) protocol enables applications to send and receive document-oriented XML messages using a pure Java API. SAAJ 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/SAAJ Web site at http://wwws.sun.com/software/communitysource/jaaxm_saaj 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

Web services architecture includes the Eclipse interface as well as a Web Management Console interface. Each supports the same functionality.

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, “Management Console—Web Services” and Chapter 6, “Management 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.”

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 28.
- Creating a Web service from a local or remote WSDL file or Java file – See “Creating Web services from files” on page 29.
- Exposing an installed EAServer component as a Web service – See “Other components” on page 44.

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:8000/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 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 not installed as part of the standard EAServer installation. To run Eclipse you must have a complete JDK installation (jdk1.4 or higher), which is installed as part of the standard EAServer installation.

❖ **Starting Eclipse in UNIX**

- From the command line in the *Shared/eclipse_311/eclipse* subdirectory, enter the command:

```
./starteclipse.sh
```

❖ **Starting Eclipse in Windows**

- From the command line in the *Shared\eclipse_311\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 26 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.

- 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 and Datatypes.”

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 which are located in the *logs* subdirectory of your EAServer installation.
- 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.

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 Hierarchy). 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 and Datatypes

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.

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.

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 and corresponding PowerBuilder types. Exposing a component as a Web service does not require you to regenerate its remote interface. EAServer uses JAX-RPC mapping rules to generate EJB remote interfaces.

Table 3-1: Supported datatypes

CORBA IDL type	Parameter mode	CORBA/Java type	EJB parameter type	PowerBuilder types
boolean	in, return	boolean	boolean	Boolean by value
	out, inout	org.omg.CORBA.BooleanHolder	javax.xml.rpc.holders.BooleanHolder	Boolean by reference
char	in, return	char	char (see note 9)	Char by value
	out, inout	org.omg.CORBA.CharHolder	N/A (see note 1)	Char by reference
octet	in, return	byte	byte	Char by value (see note 2)
	out, inout	org.omg.CORBA.ByteHolder	javax.xml.rpc.holders.ByteHolder	Char by reference (see note 2)
short	in, return	short	short	Integer by value
	out, inout	org.omg.CORBA.ShortHolder	javax.xml.rpc.holders.ShortHolder	Integer by reference
long	in, return	int	int	Long by value

CORBA IDL type	Parameter mode	CORBA/Java type	EJB parameter type	PowerBuilder types
	out, inout	org.omg.CORBA.IntHolder	javax.xml.rpc.holders.IntHolder	Long by reference
long long	in, return	long	long	LongLong by value
	out, inout	org.omg.CORBA.LongHolder	javax.xml.rpc.holders.LongHolder	LongLong by reference
float	in, return	float	float	Real by value
	out, inout	org.omg.CORBA.FloatHolder	javax.xml.rpc.holders.FloatHolder	Real by reference
double	in, return	double	double	Double by value
	out, inout	org.omg.CORBA.DoubleHolder	javax.xml.rpc.holdersDoubleHolder	Double by reference
string	in, return	string	string	String by value
	out, inout	org.omg.CORBA.StringHolder	javax.xml.rpc.holders.StringHolder	String by reference
BCD::Binary	in, return	byte[]	byte[]	Blob by value
	out, inout	BCD.BinaryHolder	javax.xml.rpc.holders.ByteArrayHolder	Blob by reference
BCD::Decimal	in, return	BCD.Decimal	java.math.BigDecimal	Decimal by value
	out, inout	BCD.DecimalHolder	javax.xml.rpc.holders.BigDecimalHolder	Decimal by reference
BCD::Money	in, return	BCD.Money	java.math.BigDecimal	Decimal by value
	out, inout	BCD.MoneyHolder	javax.xml.rpc.holders.BigDecimalHolder	Decimal by reference
MJD::Date	in, return	MJD.Date	java.util.Calendar	Date by value
	out, inout	MJD.DateHolder	javax.xml.rpc.holders.CalendarHolder	Date by reference
MJD::Time	in, return	MJD.Time	java.util.Calendar	Time by value
	out, inout	MJD.TimeHolder	javax.xml.rpc.holders.CalendarHolder	Time by reference
MJD::Timestamp	in, return	MJD.Timestamp	java.util.Calendar	DateTime by value
	out, inout	MJD.TimestampHolder	javax.xml.rpc.holders.CalendarHolder	DateTime by reference
XDT::BooleanValue	in, return	XDT.BooleanValue	java.lang.Boolean	XDT_BooleanValue by value

CORBA IDL type	Parameter mode	CORBA/Java type	EJB parameter type	PowerBuilder types
	out, inout	XDT.BooleanValueHolder	javax.xml.rpc.holders.BooleanWrapperHolder	XDT_BooleanValue by reference
XDT::CharValue	in, return	XDT.CharValue	java.lang.Character (see note 9)	XDT_CharValue by value
	out, inout	XDT.CharValueHolder	XDT.CharacterWrapperHolder (see note 1)	XDT_CharValue by reference
XDT::ByteValue	in, return	XDT.ByteValue	java.lang.Byte	XDT_ByteValue by value
	out, inout	XDT.ByteValueHolder	javax.xml.rpc.holders.ByteWrapperHolder	XDT_ByteValue by reference
XDT::ShortValue	in, return	XDT.ShortValue	java.lang.Short	XDT_ShortValue by value
	out, inout	XDT.ShortValueHolder	javax.xml.rpc.holders.ShortWrapperHolder	XDT_ShortValue by reference
XDT::IntValue	in, return	XDT.IntValue	java.lang.Int	XDT_IntValue by value
	out, inout	XDT.IntValueHolder	javax.xml.rpc.holders.IntegerWrapperHolder	XDT_IntValue by reference
XDT::LongValue	in, return	XDT.LongValue	java.lang.Long	XDT_LongValue by value
	out, inout	XDT.LongValueHolder	javax.xml.rpc.holders.LongWrapperHolder	XDT_LongValue by reference
XDT::FloatValue	in, return	XDT.FloatValue	java.lang.Float	XDT_FloatValue by value
	out, inout	XDT.FloatValueHolder	javax.xml.rpc.holders.FloatWrapperHolder	XDT_FloatValue by reference
XDT::DoubleValue	in, return	XDT.DoubleValue	java.lang.Double	XDT_DoubleValue by value
	out, inout	XDT.DoubleValueHolder	javax.xml.rpc.holders.DoubleWrapperHolder	XDT_DoubleValue by reference
XDT::DecimalValue	in, return	XDT.DecimalValue	java.lang.BigDecimal	XDT_DecimalValue by value
	out, inout	XDT.DecimalValueHolder	javax.xml.rpc.holders.BigDecimalHolder	XDT_DecimalValue by reference
XDT::IntegerValue	in, return	XDT.IntegerValue	java.math.BigInteger	XDT_IntegerValue by value
	out, inout	XDT.IntegerValueHolder	javax.xml.rpc.holders.BigIntegerHolder	XDT_IntegerValue by reference

CORBA IDL type	Parameter mode	CORBA/Java type	EJB parameter type	PowerBuilder types
XDT::DateValue	in, return	XDT.DateValue	java.util.Calendar	XDT_DateValue by value
	out, inout	XDT.DateValueHolder	javax.xml.rpc.holders.CalendarHolder	XDT_DateValue by reference
XDT::TimeValue	in, return	XDT.TimeValue	java.util.Calendar	XDT_TimeValue by value
	out, inout	XDT.TimeValueHolder	javax.xml.rpc.holders.CalendarHolder	XDT_TimeValue by reference
XDT::DateTimeValue	in, return	XDT.DateTimeValue	java.util.Calendar	XDT_DateTimeValue by value
	out, inout	XDT.DateTimeValueHolder	javax.xml.rpc.holders.CalendarHolder	XDT_DateTimeValue by reference
XDT::ByteArray	in, return	byte[]	byte[]	Blob by value
	out, inout	XDT.ByteArrayHolder	javax.xml.rpc.holders.ByteArrayHolder	Blob by reference
MyModule::MyException (exception)	raises (throws)	MyModule.MyException	MyModule.ejb.MyException	MyModule_MyException or MyException
MyModule::MyComp (interface)	in, return	MyModule.MyComp	MyModule.ejb.MyComp	MyModule_MyComp or MyComp by value
	out, inout	MyModule.MyCompHolder	MyModule.ejb.MyCompHolder	MyModule_MyComp or MyComp by reference
MyModule::MyStruct (struct)	in, return	MyModule.MyStruct	MyModule.ejb.MyStruct	MyModule_MyStruct or MyStruct by value
	out, inout	MyModule.MyStructHolder	MyModule.ejb.MyStructHolder	MyModule_MyStruct or MyStruct by reference
MyModule::MyUnion (union)	in, return	MyModule.MyUnion	MyModule.ejb.MyUnion	MyModule_MyUnion or MyUnion by value
	out, inout	MyModule.MyUnionHolder	MyModule.ejb.MyUnionHolder	MyModule_MyUnion or MyUnion by reference
MyModule::MySequence (sequence<MyElement>)	in	MyModule.MyElement[]	MyModule.ejb.MyElement[]	MyModule_MyElement[] or MyElement[] by value

CORBA IDL type	Parameter mode	CORBA/Java type	EJB parameter type	PowerBuilder types
	return	MyModule.MyElement[]	MyModule.ejb.MyElement[]	MyModule_MySequence or MySequence
	out, inout	MyModule.MySequenceHolder	MyModule.ejb.MySequenceHolder	MyModule_MyElement[] or MyElement[] by reference
MyModule::MyArray (MyElement[N]) (see note 3)	in	MyModule.MyElement	MyModule.ejb.MyElement	MyModule_MyElement[] or MyElement[] by value
	return	MyModule.MyElement	MyModule.ejb.MyElement	MyModule_MyArray or MyArray
	out, inout	MyModule.MyArrayHolder	MyModule.ejb.MyArrayHolder	MyModule_MyElement[N] or MyElement[N] by reference
TabularResults::ResultSet (see note 4)	in, return	TabularResults.ResultSet	java.sql.ResultSet	ResultSet by value
	out, inout	TabularResults.ResultSetHolder	N/A	ResultSet by reference
TabularResults::ResultSet (see note 4)	in, return	TabularResults.ResultSet[]	java.sql.ResultSet[]	ResultSet by value
	out, inout	TabularResults.ResultSetHolder	N/A	ResultSet by reference

Note

1. The ‘char’ and ‘java.lang.Charcter’ data types have no defined XML Schema mapping for EJB Web services, and should not be used as a parameter type or structure field type if you plan to expose components as Web services.
2. PowerBuilder version 10.5 introduced a Byte data type. To use the PB Char data type for backwards compatibility, run this command (once) before deployment:

   configure idl-octet-to-pb-char

   To switch back to using the PB Byte data type, run this command (once) before deployment:

   configure idl-octet-to-pb-byte
3. IDL fixed size array types have no defined XML Schema mapping for EJB Web services, and should not be used as parameter types or structure field types if you plan to expose components as Web services. Use IDL sequences types instead (Java arrays, PowerBuilder variable sized arrays).
4. The ‘ResultSet’ data type should not be used with the PB Server Plugin if you plan to expose components as Web services, because java.sql.ResultSet is not portable in EJB Web service endpoint interfaces. Use IDL sequences of structures instead (Java arrays, PowerBuilder variable sized arrays). For EAServer, EJB return type java.sql.ResultSet maps to a complex schema element that contains the result set data and the schema for the result set. The content of the XML is mapped according to the SQL/XML ANSI standard.

```
<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>
```

Using IDL parameter modes ‘inout’ and ‘out’ with TabularResults::ResultSet(s) is not supported for exposing components as Web services.

Using arrays (IDL sequences) of structures instead of result sets simplifies coding of Web service client applications since it is portable across all application servers. When you write PowerBuilder NVO methods, which do not permit using arrays as method return types, define a row structure to represent a result row, and a table structure containing an array of row structures to represent a result set.

5 Do not use IDL ‘inout’ and ‘out’ parameter modes with the PowerBuilder server plugin, because JAX-RPC holder classes are not portable in EJB remote interfaces. For EAServer, use ‘inout’ and ‘out’ parameter modes (with the exceptions listed in note 4). When PowerBuilder components are deployed, any “by reference” parameters are mapped to IDL parameter mode ‘inout’. Therefore PowerBuilder “by reference” parameters should not be used with the PowerBuilder server plugin.

6 The PowerBuilder NVO deployment option “Allow NULL values in method parameters” is not supported if you intend to expose components as web services. It is also not supported when using the PB Server Plugin (see note 10).

7 The default mapping of CORBA IDL identifiers to Java/EJB identifiers can be modified to use Java naming conventions. This is called the “camel case” deployment option. When using this option, IDL operation and parameter names such as “abc_xyz” map to “abcXyz”, and IDL interfaces, sequence, structure, and union type names “abc_xyz” map to “AbcXyz”. This mapping is not applied to exception and structure field names.

To enable the “camel case” option, use this command:

```
configure camel-case-on
```

To disable the “camel case” option, use this command:

```
configure camel-case-off
```

If you intend to expose components as Web services, you should enable the "camel case" option, otherwise you might encounter problems with the JAX-RPC identifier mapping rules (See the JAX-RPC 1.1 specification, chapter 20 "Appendix: Mapping of XML Names").

8 For CORBA C++ data types, see the CORBA IDL to C++ Language Mapping document at http://www.omg.org/technology/documents/formal/c++.htm

9 Only characters in the ISO 8859-1 character set can be used in this case. Use the String type to propagate other characters.

10 To obtain the PowerBuilder XDT_* data types for use as PB structure field types or component parameter types, use the "EAServer Proxy Wizard" or "Application Server Proxy Wizard" in the PowerBuilder IDE to generate proxies for the "XDT" package. Each of the XDT_* data types contains a value field and an isNull field. The isNull field must be set to true to indicate a null value.

11 Exposing PowerBuilder components as Web services where the component passes an array of simple types by reference is not supported.

Exposing CORBA Components as Web services where the component passes a sequence of simple types as inout parameters is not supported.

Instead you should pass IDL sequences of user-defined structure types using IDL inout or out parameters. Also see note 7.

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.

CHAPTER 4

Web Services Administration

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

Introduction

The WST development tool supports top-down (creating a Web service from the WSDL) and bottom-up (creating a Web service from a component) 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. `8000` 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 or JAR 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.

❖ **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 38 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 top-down 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 bottom-up 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 32 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 32 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 42.
	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 38.
	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.

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.

❖ **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 7, "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.

Window	Property	Description
	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.
	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.

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).

Do not use overloaded methods or properties in PowerBuilder components that you want to expose as Web services, or the Web service fails to be exposed.

components they want to use as web services.

❖ **Invoking an operation**

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

- Right-click the operation and then select Invoke.

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.

Property type	Property	Description
	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 and Datatypes.”

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.

❖ **Using the Expose 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

Property	Description
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.

Property	Description
Context path	Location of the Web service.
Endpoint address URI	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.
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.
Use	Specify the use (LITERAL or ENCODED) of items in the generated WSDL binding when exposing a Web service.

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.

Window	Property	Description
	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_1:8000/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.

Window	Property	Description
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.

Window	Property	Description
	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.
	Get All Business Details 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.

Window	Property	Description
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.
	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.

See “Exposing Components as Web services” on page 38 and “Using the quickly expose wizard” on page 39 for information about deploying other components as Web services.

CHAPTER 5

Management Console—Web Services

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

Plug-in, domain, display, and server administration

This section describes how to use the management 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 management console wizards, nodes, and the interface are displayed.

❖ 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 Plugins 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 47 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 47 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 47 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.

- Refresh Node– refreshes the server and any changes since the last refresh or restart.

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.
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, `8000`.
User ID	The user name used to connect to the server. `admin@system` is the default. Use `admin` or another member of the Admin role to connect to the Web services container for access to all of management console’s functions.
Password	The password of the user connecting to the server.
Auto Connect on Console Login	Select this box to connect to this profile automatically when you log in to the management console.

A node can be a plug-in, domain, 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 management 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 management console displays General and Web Service tabs. Table 5-2 on page 48 describes the Web service collection properties.

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

❖ **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 49 for a description of the Web service properties.

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

❖ **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 management console displays only the first method of the overloaded method.

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 50 for a description of the Web service properties.

❖ **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.

Property type	Property	Description
	Use	Indicates whether the message parts are encoded using some encoding rules, or whether the parts define the concrete schema of the message. If use is *encoded*, then the *Encoding Style* specifies the encoding style to be applied.
	Encoding Style	Specify the SOAP encoding style. Each encoding style is identified using a list of URIs. For example, http://schemas.xmlsoap.org/soap/encoding/ identifies SOAP encoding as defined by the SOAP specification.
	Return Type	Specifies the return type of the method.
	Return Name	Specifies the name of the return type.
	Is Return Value In Response	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 parameters**

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

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.
	Get All Business Details 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 and Datatypes.”

Managing security realms

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 Management 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 Management Console, you must refresh the realm to see those changes.

- Right-click the security realm and select Refresh.

Non-Web service components

The Non-Web Service Components folder contains components that are hosted on the server to which the management 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 38 describes the properties. When you click Finish, the Web service is exposed in the Web service collection you entered.

CHAPTER 6

Management console—Registry Services

This chapter describes how to use the Sybase Management Web Management Console to administer information contained in the local UDDI servers, and publish to a UDDI registry.

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

Introduction

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

- An administration console for the local UDDI servers – Sybase provides local UDDI registries as part of Web services. The local 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. the UDDI server is deployed as a Web application and works as any other Web application in EAServer.
- A browser capable of searching and publishing to any UDDI registry.

Using the management console

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

Navigating the console and managing resources

Navigate the management 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:

- Predefined registries - EAServer contains these predefined registries:
  - UDDI on Localhost – a UDDI registry.
  - UDDI on TrySybase – a UDDI registry.
  - WSDP registry server – a Java Web Services Developer Pack (WSDP) registry server that implements version 2 of the (UDDI) to provide a registry for Web services in a private environment. You can use it with the Java WSDP APIs as a test registry for Web services application development. For more information see:
    - The Java WSDP tutorial at http://java.sun.com/webservices/docs/1.6/tutorial/doc/
    - The Java WSDP API specification at http://java.sun.com/webservices/docs/1.6/api/index.html
- UDDI server – part of EAServer, but not installed or deployed by default. To deploy the server, use the command:

  ```
  deploy.bat ..\extras\juddi\juddi.war
  ```

  When you deploy *juddi.war* from the *$DJC_HOME/extras* directory, two users are created: juddipublish@default and juddiadmin@default. These users do not have passwords set initially. To use them for connecting and managing the UDDI registry, you must first establish a password using the set-password command. For example, from the *bin* subdirectory of your EAServer installation, enter:

  ```
  set-password
  ```

  You are prompted for a Username. Enter `juddipublish@default`. You are prompted for a password. Enter a password for this user. Enter the password a second time. You can now connect to the UDDI registry using juddipublish@default.

Once deployed, restart EAServer to access the UDDI server. You can then use any local Database for maintaining the registry information for the UDDI server.

The configuration settings for the UDDI server are located in *config\webapp-juddi.xml* and *deploy\webapps\juddi\WEB-INF\juddi.properties* files located in the EAServer installation subdirectory.

- Registry Administration – includes defining and managing registries. See "UDDI registry profile administration" on page 60.
- Search – search UDDI registries. See "Inquiries and searches" on page 61.
- Publish – publish business information to UDDI registries. See "Publishing" on page 63.

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 management 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 management 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 61.

❖ **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 management console attempts to connect to the registry with the information provided when it was created. See Table 6-1 on page 61. If the management 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 profile name	The name of the registry you are creating or modifying.
	Query URL	The location from which you query the UDDI.
	Publish URL	To publish, you need both query and publish URLs.
	User name	The user name used for accessing the UDDI site. The default is admin@system
	Password	The password used in connection with the user name used to access the UDDI site.
	Auto connect to the registry server.	Select this box to connect to this registry automatically when you log in to the management console
	Ping Registry	Verifies that the information provided allows connection to the registry.
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. Before you can search or publish to a registry, you must be connected to it. See “Connecting to a UDDI registry profile” on page 60.

Inquiries and searches

From the management 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 for the registry to which you are connected.

2 Click the desired tab, and 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 67 for more information about categories.
	Add Identifier	Add an identifier to this business. See “Identifiers” on page 68 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 67 for more information about categories.

Search type	Options	Description
tModel	tModel name	Enter a name of a tModel for which you are searching. **Note** When performing a tModel search against either the UDDI on TrySybase, or UDDI on Localhost, the search is always performed in a case sensitive manner regardless of the case sensitive setting on the search page.
	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 67 for more information about categories.
	Add Identifier	Add an identifier to this business. See “Identifiers” on page 68 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 management 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 64 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 64
- Contact – see “Contacts” on page 69
- Discovery URLs – see “Discovery URLs” on page 70
- Categories – see “Categories” on page 67
- Identifiers – see “Identifiers” on page 68

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 65 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 66
- Categories – see “Categories” on page 67
- Identifiers – see “Identifiers” on page 68

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 66 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 70
- Categories – see “Categories” on page 67
- Identifiers – see “Identifiers” on page 68

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 67 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 68 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 6-7 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 69 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 70 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 70 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

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 74
- “Dynamic proxy client” on page 74
- “Dynamic invocation interface client” on page 75
- “Document style client” on page 75

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 33. 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
```

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 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.

The significant JAX-RPC client APIs used are:

- javax.xml.rpc.Call
- javax.xml.rpc.Service
- javax.xml.rpc.ServiceFactory

Document style client

The previous clients require 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.

CHAPTER 8

J2EE Web Service Support

Overview

Web service support in EAServer 6.0 includes the ability to deploy J2EE Web applications and EJBs as Web Services. J2EE Web Services support is new in EAServer 6.0, and described in this chapter.

J2EE Web services support

This section provides an overview of J2EE Web services support included in EAServer 6.0, including:

- Web Services Runtime Support – Web Services runtime includes a Web Service container that supports servlet style Web services. EJBs are bundled as Web services and deployed as servlets at deployment time.
- Web Service Client `<service-ref>` support – When a Web service client (EJB or servlet) is deployed the `<service-ref>` is requested, the service-interface class is returned to the client with which to work.
- Application client and EJB client support – client reference support for EJBs and application clients is similar to that of Web applications. All three have a `service-ref` definition, and JNDI lookups in common. The `service-ref` definitions are contained in the J2EE `-entitytype` xml file; *web.xml* for webapp, *ejb-jar.xml* for EJB JAR, and *application-client.xml* for application clients.

- UDDI Server - an Apache UDDI server (jUDDI) is included along with custom configuration code. An installer option is available for deployment. The deployable WAR file is *EAServer_home/extras/juddi/juddi.war*.

  Deploying the *juddi.war* file sets up the UDDI server and creates the appropriate tables in the default data source.

For additional information, refer to the J2EE 1.4 specification, at the Sun Developer Network at http://java.sun.com/j2ee/1.4/index.jsp.

Deploying J2EE Web services

You can deploy Web services that are contained in J2EE archive files by deploying the archive file, including:

- EJB Jar files – can contain J2EE 1.4 EJB Web services.
- WAR Files – can contain J2EE 1.4 Web application (servlet based) Web services.
- EAR files – can contain EJB Jar files or WAR files that contain J2EE 1.4 Web services.

A Web service within one of these archive files is defined by a combination of the *webservices.xml* file, WSDL file, and *jaxrpcmapping* file.

The *webservices.xml* file defines the location of the WSDL file and *jaxrpcmapping* file, and which EJB/servlet is used to define the Web service.

Additional information about the *webservices.xml* file can be obtained from it's XSD, as described by these Web sites:

- J2EE deployment descriptors at http://java.sun.com/xml/ns/j2ee and
- J2EE Web service xsd documentation at http://java.sun.com/xml/ns/j2ee/j2ee_web_services_1_1.xsd

You can deploy Web service clients as part of an EJB, Web application, or application client. These are defined by a `service-ref` tag in the *ejb-jar.xml*, *web.xml* or *application-client.xml* file.

Web Service clients can be deployed with full WSDL, partial WSDL, or no WSDL as defined in the J2EE 1.4 Web Services specification. If you deploy with partial or no WSDL, additional information must be given on deployment for port name, binding and location address. See "Deploying with a partial WSDL" on page 84.

❖ **Deploying J2EE Web services**

1 Deploy your J2EE Web service from the command line.

2 Go to the bin subdirectory of your EAServer installation.

3 Use the deploy command (*EAServer_home/bin/deploy.bat* Windows *deploy.sh* Unix) to deploy the EJB Web service. See "Deploying Web services from the command line" on page 81

Viewing Web services

After deploying Web services to EAServer you can view the contents.

❖ **Viewing deployed Web application (servlet) Web service**

1 From the Web Management Console, expand the Web Applications folder.

2 Highlight the Web application you want to view.

3 Select the *web.xml* tab. You can view Web service client `<service-ref>` information that provides the Web service binding references for the Web service. For example:

```
<service-ref>
   <service-ref-name>service/service</service-ref-name>
   <service-
interface>com.sun.ts.tests.webservices.deploy.GenSvc.TestsGenSvc
   </service-interface>
   <wsdl-file>META-INF/wsdl/TestsGenSvc.wsdl</wsdl-file>
   <jaxrpc-mapping-file>TestsGenSvc.xml</jaxrpc-mapping-file>
   <port-component-ref>
      <service-endpoint-interface>
      com.sun.ts.tests.webservices.deploy.GenSvc.Tests
      </service-endpoint- interface>
   </port-component-ref>
</service-ref>
```

Some main features of the client reference are:

- `<service-ref-name>` is used in a JNDI lookup to retrieve an instance of the <service-interface>.

- `<jaxrpc-mapping-file>` is also used during deployment to map namespaces to Java packages.
- `<service-endpoint-interface>` is the actual interface you use to call your business methods.

This example illustrates how you can use the myecho object to call methods on the service endpoint interface String rc = myecho.echo("Hello remote world");:

```
EchoService myservice =
(EchoService)context.lookup("java:comp/env/service/EchoService")
// This gives the client the service-interface. From which you can look
up the service-endpoint-interface as follows:
Echo myecho = myservice.getEcho(); or
Echo myecho = (Echo)myservice.getPort(Echo.class);
```

4 Use the deploy command (*EAServer_home/bin/deploy.bat*) to deploy a J2EE 1.4 WAR file containing a Web service or Web service client.

❖ **Viewing EJB Web services**

1 From the Web Management Console, expand the EJB Modules folder.

2 Highlight the EJB you want to view.

3 Select the *ejb-jar.xml* tab. View Web service client `<service-ref>` information. See "Viewing deployed Web application (servlet) Web service" on page 79 for an example.

4 Use the deploy command (*EAServer_home/bin/deploy.bat*) to deploy the EJB Web service.

❖ **Viewing application client Web services**

1 From the Web Management Console, expand the Application Client folder.

2 Highlight the application client you want to view.

3 Select the *application-client.xml* tab. View Web service client `<service-ref>` information. See "Viewing deployed Web application (servlet) Web service" on page 79 for an example.

4 Use the deploy command (*EAServer_home/bin/deploy.bat*) to deploy the Web service.

Deploying Web services from the command line

This section describes the command-line options for deploying J2EE Web services using the deploy command located in the *bin* subdirectory of your EAServer installation. See Chapter 12, “Command Line Tools” for more information about this, and other command line tools.

Command line:

```
deploy
[options]
entity
[-contextpath path]
```

The most simple form of the deploy command for the various archive files is:

- deploy *foo.jar*
- deploy *foo.ear*
- deploy *foo.war*

Where:

Option	Description
`-ws`	There are three ws options that define how a Web service is exposed: • -ws – expose any Stateless Session Beans with a Remote Interface as a Web service. • -ws:<*ejbName*> – expose the Stateless Session Bean *ejbName* as a Web service. • -ws:<*jarFileInEar*>:<*ejbName*> – expose the Stateless Session Bean *ejbName* located in an application EAR file *jarFileInEar* as a Web service.
`-wsClientAddress`	Use this syntax to overwrite the <soap:address location> in the WSDL file referred to by <service-ref-name> in the *web.xml*, *ejb-jar.xml* or *application-client.xml* and the <port name> in the WSDL file: -wsClientAddress:<*serviceRefName*>:<*portComponentName*> <*address*>
`-wsContextPath` <contextpath>	Use this syntax to specify the context path for an EJB Web service in an application EAR file: -wsContextPath:<*jarFileInEar*>:<*contextpath*> Use this syntax to specify the context path for an EJB Web Service: -wsContextPath <*contextpath*>

Option	Description
`-wsEndpointAddressURI`	Specify the endpoint address URI for an EJB Web service using this syntax: -wsEndpointAddressURI <*ejbName*>:<*endpoint-address-uri*> Specify the endpoint address URI for an EJB Web service in an Application EAR file using this syntax: -wsEndpointAddressURI:<*jarFileInEar*>:<ejbName>: <*endpoint-address-uri*>
`-wsStyle`	Specify the Web service style (DOCUMENT, RPC or WRAPPED) of the generated WSDL binding used when exposing an EJB as a Web service using this syntax: -wsStyle <*style*> Specify the Web service style (DOCUMENT, RPC or WRAPPED) of the generated WSDL binding used when exposing an EJB as a Web service contained in an application EAR file using this syntax: -wsStyle:<*jarFileInEar*>:<*style*>
`-wsUse`	Specify the use (LITERAL or ENCODED) of items in the generated WSDL binding when exposing an EJB as a Web service using this syntax: -wsUse <*use*> Specify the use (LITERAL or ENCODED) of items in the generated WSDL binding when exposing an EJB as a Web service in an application EAR file using this syntax: -wsUse:<*jarFileInEar*>:<*use*>
`-wsWebAppName`	Override the default name for the Web application generated from an EJB Web Service using this syntax: -wsWebAppName <*webappname*> Override default name for the Web application generated from an EJB Web Service in an application EAR file using this syntax: -wsWebAppName:<*jarFileInEar*>:<*webappname*>
`-entity`	The file that you are deploying. *entity* should be located in the current directory, or provide the full path using the contextpath option.

Option	Description
`-contextpath` path	Specify the contextPath for Web application deployment. The default is the name of the WAR file. If -package is specified, then the package is the context path.
`-help`	Enter deploy -help to display all command line options

Examples

This example uses the deploy and undeploy commands located in EAServer's *bin* subdirectory to demonstrate using default context path and default end point addresses for stateless EJB Web Services by deploying a Web service contained in the *HiWS.jar* file, and the client that is contained in the *HiWSClient.war* file.

1 Deploy the *HiWS.jar* file:

```
deploy.bat C:\WebSvcSample\setA\HiWS.jar (Windows)
deploy.sh WebSvcSample/setA/HiWS.jar (Unix)
```

2 Verify the EJB Web Service URL location from your browser:

```
http://localhost:8000/hiws/HiWS?WSDL
```

The WSDL of HiWS EJB Web service displays

3 Deploy the *HiWSClient.war* file by entering this command from EAServer's *bin* subdirectory:

```
deploy.bat C:\WebSvcSample\setA\HiWSClient.war (Windows)
deploy.sh WebSvcSample/setA/HiWSClient.war (Unix)
```

4 Test the servlet by entering this URL in your browser:

```
http://localhost:8000/HiWSClient/HiServlet
```

Enter a name (e.g., John) and click Enter

Result: Hi John!

5 Undeploy the EJB and Web applications by entering these commands from EAServer's *bin* subdirectory

```
undeploy ejbjar-hiws
```

```
undeploy webapp-hiwsclient
```

This example uses the deploy command to specify the context path and end point address for a stateless EJB Web service by matching the context path and end point address to the SOAP address specified in the *HiWSClient.war*'s *hiWS.wsdl* file. This typically happens when one organization provides the Web service and another organization uses/consumes the Web service. The set of published URIs (e.g., webservice/sayHi) serves as a contract between Web service provider and Web service consumer.

1 Deploy the *HiWS.jar* file:

```
deploy.bat -wsContextPath webservice -wsEndpointAddressURI
HiWS:sayHi C:\WebSvcSample\setB\HiWS.jar
```
(Windows)

```
deploy.sh -wsContextPath webservice -wsEndpointAddressURI
HiWS:sayHi WebSvcSample/setB/HiWS.jar
```
(Unix)

2 Verify the EJB Web service URL location by entering this in your browser:

```
http://localhost:8000/\b webservice/sayHi\b0 ?WSDL
```

The WSDL of HiWS EJB Web service displayes

3 Deploy the *HiWSClient.war* file:

```
deploy.bat C:\WebSvcSample\setB\HiWSClient.war
```
(Windows)

```
deploy.sh WebSvcSample\setB\HiWSClient.war
```
(Unix)

4 Test the servlet by entering this URL in your browser:

```
http://localhost:8000/HiWSClient/HiServlet
```

Enter a name (e.g., John) and click Enter

Result: Hi John!

5 Undeploy the EJB and Web applications:

```
undeploy ejbjar-hiws
```

```
undeploy webapp-hiwsclient
```

Deploying with a partial WSDL

In accordance with the J2EE 1.4 specification, you can occasionally deploy without a complete WSDL file. If done, you must complete some of the service definition. As the following examples illustrate.

Changing the location (SOAP address) for the client:

```
<webServiceRef configName="${this.config.name}" package="${this.package.name}"
serviceRefName="service/HiWS" merge="false">
   ...other props
   <service name="HiWS">
   <port name="HiWSSEIPort"
   location="http://mymachine:8000/it/worked"/>
   </service>
   ...
</webServiceRef>
```

Completing a partial or incomplete WSDL file:

If your WSDL file does not define a service name, you need to specify service name, port name, binding, and location. For example:

```
<webServiceRef configName="${this.config.name}" package="${this.package.name}"
serviceRefName="service/HiWS" merge="false">
      ... other props
    <property name=wsdlLocation" value="c:/myspot/mywsdl.wsdl"/>
    <service name="HiWS">
        <port name="HiWSSEIPort" binding="tns:HiWSSEIBinding"
location="http://mymachine:8000/it/worked"/>
    </service>
      ...
</webServiceRef>
```

This example is for a Web application that contains a Web service client with some WSDL completion values:

```
<?xml version="1.0"?>
<project name="webapp-hiwsclient" default="configure">
  <property name="this.config.name" value="webapp-hiwsclient"/>
  <import file="ant-config-tasks.xml"/>
  <import file="default-config-targets.xml"/>
  <property name="this.package.name" value="hiwsclient"/>
  <import file="${this.config.name}-user.xml" optional="true"/>
  <property name="djc.verbose" value="false"/>
  <property name="web.accessControl" value="default"/>
  <property name="web.allowedPorts" value="all"/>
  <property name="web.rolePrefix" value="hiwsclient"/>
  <property name="web.contextPath" value="HiWSClient"/>
  <property name="web.virtualHost" value=""/>
  <property name="web.logExceptions" value="true"/>
  <property name="web.enableProfiling" value="true"/>
  <property name="web.enableTracing" value="true"/>
  <property name="web.threadMonitor" value="default"/>
  <property name="web.javacTarget" value="1.4"/>
```

```
  <property name="web.deployDir"
value="${djc.home}/deploy/webapps/hiwsclient"/>
  <property name="web.classDir" value="${web.deployDir}/WEB-INF/classes"/>
  <property name="web.compileJspDir" value="${web.deployDir}/WEB-
INF/compiled_jsps"/>
  <path id="web.classpath.path.id">
    <pathelement path="${web.classDir}"/>
   <!-- WEB-INF/lib may not be present in the war, ANT will complain if lib is
included in the dir= attribute-->
    <fileset dir="${djc.home}/deploy/webapps/hiwsclient/WEB-INF"
includes="lib/*.jar lib/*.zip" casesensitive="no"/>
    <fileset dir="${djc.home}/lib/default/ext" includes="*.jar *.zip"
casesensitive="no"/>
    <fileset dir="${djc.home}/lib/ext" includes="*.jar *.zip"
casesensitive="no"/>
  </path>
  <pathconvert pathsep="${path.separator}" property="web.classPath"
refid="web.classpath.path.id"/>
  <property name="jca.connectionFactory" value="default"/>
  <property name="sql.dataSource" value="default"/>
  <target name="configure-default">
    <echo level="info" message="configure: webapp-hiwsclient"/>
    <setProperties component="web.components.hiwsclient.HiServlet"
merge="false">
      <threadMonitor name="${web.threadMonitor}"/>
      <transaction type="BeanManaged"/>
    </setProperties>
    <setProperties component="web.components.hiwsclient.JspServlet"
merge="false">
      <threadMonitor name="${web.threadMonitor}"/>
      <transaction type="BeanManaged"/>
    </setProperties>
    <setProperties package="web.components.hiwsclient" merge="false">
      <property name="contextPath" value="${web.contextPath}"/>
      <property name="virtualHost" value="${web.virtualHost}"/>
      <accessControl type="${web.accessControl}"/>
      <logExceptions enable="${web.logExceptions}"/>
      <profilePublicMethods enable="${web.enableProfiling}"/>
      <tracePublicMethods enable="${web.enableTracing}"/>
      <classLoader name="web.components.hiwsclient"/>
      <permitAccess ports="${web.allowedPorts}"/>
      <property name="rolePrefix" value="${web.rolePrefix}"/>
      <!-- WebServiceRef: java:comp/env/service/HiWS-->
      <bind name="java:comp/env/service/HiWS"
webService="web.components.hiwsclient.service.HiWS"/>
    </setProperties>
```

```
    <webServiceRef configName="${this.config.name}"
package="${this.package.name}" serviceRefName="service/HiWS" merge="false">
      <property name="archiveFile" value="HiWSClient.war"/>
      <property name="deploymentDescriptorFile"
value="M:\target1.4\deploy\webapps\hiwsclient\WEB-INF\web.xml"/>
      <property name="deploymentDescriptorType" value="webapp"/>
      <property name="wsdlLocation" value="~/deploy/webapps/hiwsclient/WEB-
INF/wsdl/HiWS.wsdl"/>
      <property name="localWsdlLocation"
value="~/deploy/webapps/hiwsclient/WEB-INF/wsdl/HiWS.wsdl"/>
      <service name="HiWS">
        <port name="HiWSSEIPort" binding="tns:HiWSSEIBinding"
location="http://mymachinename:8000/it/worked"/>
      </service>
      <property name="serviceInterface" value="org.me.hi.HiWS"/>
      <property name="jaxrpcMappingFile"
value="~/deploy/webapps/hiwsclient/WEB-INF/HiWS-mapping.xml"/>
      <property name="serviceName" value="HiWS"/>
    </webServiceRef>
    <setProperties classLoader="web.components.hiwsclient" merge="false">
      <property name="classPath" value="~/deploy/webapps/hiwsclient/WEB-
INF/classes;~/deploy/webapps/hiwsclient/WEB-INF/lib/**"/>
      <property name="resolveFirstBySystem"
value="org.apache.commons.logging.**,javax.xml.parsers.**, org.w3c.dom.**,
org.xml.sax.**"/>
      <property name="parentFirst" value="false"/>
      <property name="parentClassLoader" value="lib.default-ext"/>
    </setProperties>
  </target>
  <target name="recompile-default">
    <echo level="info" message="recompile: webapp-hiwsclient"/>
    <rewriteWsdlAddress webService="web.components.hiwsclient.service.HiWS"/>
    <djc package="web.components.hiwsclient"/>
    <javac target="${web.javacTarget}" source="${web.javacTarget}"
srcdir="${djc.home}/genfiles/java/src" destdir="${web.classDir}"
      classpath="${web.classPath}"
      includes="web/components/hiwsclient/**"/>
  </target>
  <target name="refresh-default">
    <echo level="info" message="refresh: webapp-hiwsclient"/>
    <refresh module="webapp-hiwsclient"/>
  </target>
  <target name="deploy-default">
    <echo level="info" message="deploy: webapp-hiwsclient"/>
  </target>
  <target name="undeploy-default">
```

```
    <echo level="info" message="undeploy: webapp-hiwsclient"/>
    <delete
file="${djc.home}/Repository/Instance/com/sybase/djc/util/DjcClassLoader/web.
components.hiwsclient.properties"/>
    <delete
file="${djc.home}/Repository/Instance/com/sybase/djc/ws/client/WebService/web
.components.hiwsclient.service.HiWS.properties"/>
    <unload module="webapp-hiwsclient"/>
  </target>
</project>
```

It is also possible to point at a different WSDL file, perhaps one with corrected addresses or ports. The property for this is:

```
<property name="wsdlLocation" value="<wsdllocation>"/>
```

For example:

```
<property name="wsdlLocation" value="~/deploy/appclients/ggl5497.usv8202/META-
INF/wsdl/TestsServicePartial.wsdl"/>
```

Note ~deploy refers to the servers deploy directory.

Stub properties

It is possible to set up stub properties at deployment time, the format is:

```
Stub Properties:  (use the portComponent tag which
specifies either serviceEndpointInterface or wsdlPort
or both and then the stubProperty tag with name/value
pairs for the stub properties)

<webServiceRef configName="${this.config.name}"
package="${this.package.name}"

ejbName="com_sun_ts_tests_common_vehicle_ejb_EJBVehicl
e" serviceRefName="service/handlersec">
      ...
      <portComponent
serviceEndpointInterface="com.sun.ts.tests.webservices
.handler.HandlerSec.TestAuth"
                     wsdlPort="TestAuthPort">
        <stubProperty
name="javax.xml.rpc.security.auth.password"
value="javajoe"/>
        <stubProperty
```

```
name="javax.xml.rpc.security.auth.username"
value="javajoe"/>
      </portComponent>
      ...
</webServiceRef>
```

The two are both usable depending on whether you associate the stub property with the service endpoint interface or WSDL port.

Setting the EJB Web service Web application suffix

You can change the default EJB Web service Web application package suffix in *%DJC_HOME%/config* by modifying this section of the *deploy-tool-options.xml* file:

```
<!-- General Deployment Properties -->
<setProperties component="com.sybase.djc.deploy.DeployTool" merge="true">
   <property name="disableValidation" value="false"/>
   <property name="jacc" value="false"/>
   <property name="keepModuleOnFailure" value="false"/>
   <property name="overwrite" value="true"/>
   <property name="wspackagesuffix" value=""/>
</setProperties>
```

Change the value for the property wspackagesuffix to change the EJB Web service generated Web application name. For example, if the value is "_myservice", and you deploy an EJB in an EJB JAR file called *MyEjb.jar*, the resulting Web application is called *MyEjb_myservice*.

Web service file locations and access points

This section describes where the configuration and WSDL files are stored for your generated J2EE 1.4 Web services, and where the access points are for those Web services.

There are three types of components which generate J2EE 1.4 artifacts:

- PowerBuilder components – generate EJBs, which generate Web applications which are then deployed as a Web application with Web services. The Web service is called which references the Web application, which references the EJB, which references the PowerBuilder component. See “A PowerBuilder component deployed/exposed as a Web service” on page 90.
- EJBs – generate Web applications which are then deployed as a Web application with Web services. The Web service is called which references the Web application which references the EJB. See “An EJB exposed/deployed as a Web service” on page 91.
- Web Applications – deployed directly as Web services. The Web service is called which references the Web application. See “A Web application deployed as a Web service” on page 92.

Note The `wsPackageSuffix` property in the *deploy-tools-options.xml* file controls Web application suffix naming. By default it is ““, that is there is no suffix. changing this property to something else results in the `wsPackageSuffix` being appended to the name of the `wsWebApp`, for example:

```
<property name="wsPackageSuffix" value="_webService"/>
```

results in a Web application name of `myejbjar_webService`.

A PowerBuilder component deployed/exposed as a Web service

When you expose your PowerBuilder component as a Web service from your IDE (Integrated Development Environment), an EJB is created from the PowerBuilder component. The deployed EJB ultimately has a J2EE Web services description file associated with it.

The EJB uses the information contained in the Web services description file to generate:

- A Web application which contains a servlet – located in the *deploy\webapps\WS_name* subdirectory of your EAServer installation, where *WS_name* is the name of the Web application. It contains all the relevant Web service files including configuration files and the original and modified WSDL files. The modified WSDL file contains the actual address of your Web service. For example:

```
<wsdlsoap:address location="http://mymachine:8000/pbsoap_ws/n_pbsoap"/>
```

where *pbsoap_ws* is the name of the Web service and *n_pbsoap* is the name of the PowerBuilder component you exposed as a Web service.

- Associated files for the EJB – located in the *deploy\ejbjars\EJB_name* subdirectory of your EAServer installation, where *EJB_name* is the name of the EJB.

❖ **An example of exposing a PowerBuilder component as a Web service**

1. You have a component (NVO) named n_pbsoap with a package name of pbsoap in your PowerBuilder IDE.
2. You expose the component from your IDE using these comments:

```
javaPackage="com.sybase.mypackage";webServices="n_pbsoap";
```

3. A Web application is deployed to the *deploy\webapps\pbsoap_ws* subdirectory of your EAServer installation.
4. Access the Web service WSDL at
   `http://mymachine:8000/pbsoap_ws/n_pbsoap?wsdl`

An EJB exposed/deployed as a Web service

Exposing an EJB is very similar to exposing a PowerBuilder component, except that:

- A Web application generated from an EJB has the name *myejbjar*, where *myejbjar* was the name of your EJB Jar file.
- The *myejbjar* file is deployed to the *deploy\webapps\myejb* subdirectory of your EAServer installation.
- The WSDL can be accessed at
  `http://mymachine:8000/myejbjar/myejbname?wsdl`
- When generating Web services from EJBs, these configuration files are generated:
  - *ws-ejbjarname-ejbname.xml* is generated for the EJB
  - *webapp-ejbjarname.xml* is generated for the Web application.

  For example, if you deploy an EJB with an EJB Jar name of *myejbjar*, these files are created in the *config* subdirectory of your EAServer installation:
  - *ejbjar-myjarname.xml*

- *ejbjar-myjarname-user.xml*
- *webapp-ejbjarname.xml*
- *ws-ejbjar-ejbjarname.xml*

A Web application deployed as a Web service

When you deploy a Web application that contains a Web service, the configuration and WSDL files are deployed to the *webapps\mywebapp* subdirectory of your EAServer installation, where *mywebapp* is the name of the Web application. The WSDL is available at

```
http://mymachine:8000/mywebapp/myservletname?wsdl
```

CHAPTER 9

Using wstool and wstant

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).

Working with wstool

Before using wstool, make sure that the DJC_HOME environment variable is set to the EAServer installation directory. Use the following script to run wstool:

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

wstool syntax

The syntax for wstool is:

wstool [*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 the server on which EAServer resides
`-n` or `-port`	Web services host port number; default is 8080
`-u` or `-user`	User name; default is admin@system
`-p` or `-password`	Password; default is the password you established during installation of EAServer. You can change this password using the set-admin-password command.
`-k` or `-protocol`	Communication protocol; default is "http"
`-l` or `-logfile`	Log file name; default is "System.out"

- *command* is a wstool command.

For example, to connect to the server running on "paloma" at HTTP port "9005", using account "admin@system" with password "1secret" enter:

```
wstool -h paloma -n 9005 -u admin@system -p 1secret
```

Note wstool command options are not case sensitive.

Return codes

wstool 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 command by using the help option. For example to display all of the wstool commands, enter:

```
wstool 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
```

Verbose

All wstool 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 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 9-1 provides examples of entity identifiers for each entity type.

Table 9-1: Example entity identifiers

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

Not all wstool 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

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

Before running wstant, verify that:

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

  By default, wstant searches for Jakarta Ant in *%DJC_HOME%\jakarta-ant* (Windows) or *$DJC_HOME/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 scripts.

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

wstant scripts

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

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

wstant syntax

wstant scripts uses this syntax:

wstant *[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.

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 98
- Server management commands on page 102
- Web service administration commands on page 107

UDDI administration commands

Description

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

Command list

Table 9-2 lists the UDDI administration commands described in this section.

Table 9-2: 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.

Option	Description
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 TrySybase registry:

```
wstool inquiry -inquiryURL http://uddi.trysybase.com:8080/uddi/inquiry
-business myBusiness
```

Ant build example:

```
<wst_antTask command="inquiry"
inquiryURL="http://uddi.trysybase.com:8080/uddi/inquiry"
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 TrySybase registry:

```
wstool publish -inquiryURL http://uddi.trysybase.com:8080/uddi/inquiry
-publishURL http://uddi.trysybase.com:8080/uddi/publish -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="http://uddi.trysybase.com:8080/uddi/inquiry"
publishURL="http://uddi.trysybase.com:8080/uddi/publish" 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 TrySybase registry:

```
wstool unpublish -inquiryURL http://uddi.trysybase.com:8080/uddi/inquiry
-publishURL http://uddi.trysybase.com:8080/uddi/publish -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="http://uddi.trysybase.com:8080/uddi/inquiry"
publishURL="http://uddi.trysybase.com:8080/uddi/publish" 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 9-3 lists the server management commands.

Table 9-3: 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]
[ServerProps]
[ServerVersion]
[ServiceName]
[Services]
[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"
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*

Type	Description
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*
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*
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*
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

Type	Description
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 9-4 lists the Web service administration commands.

Table 9-4: 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.
deploy (4)	Command-line deployment options for J2EE Web services.
disallowMethods	Makes Web service methods unavailable to Web service clients.
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.
refresh	Refreshes a server or Web service collection.
set_props	Sets the value of the property for a component, Web application, or a 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.

Option	Description
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"/>
```

deploy (4)

Description

These are the command-line options for deploying J2EE Web services.

Syntax

Command line:

```
deploy
[options]
entity
[-contextpath path]
```

Ant build file:

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

Where:

Option	Description
ws	There are three ws options that define how a Web service is exposed: • -ws – expose any Stateless Session Beans with a Remote Interface as a Web service. • -ws:<*ejbName*> – expose the Stateless Session Bean *ejbName* as a Web service. • -ws:<*jarFileInEar*>:<*ejbName*> – expose the Stateless Session Bean *ejbName* located in an application EAR file *jarFileInEar* as a Web service.
wsClientAddress	Use this syntax to specify the address (in the WSDL file) to which a client is referring: -wsClientAddress:<*serviceRefName*>: <*portComponentName*> <*address*>
wsContextPath	Use this syntax to specify the context path for an EJB Web service in an application EAR file: -wsContextPath:<*jarFileInEar*>:<*contextpath*> Use this syntax to specify the context path for an EJB Web Service: -wsContextPath <*contextpath*>

Option	Description
wsEndpointAddress URI	Specify the endpoint address URI for an EJB Web service using this syntax: -wsEndpointAddressURI <*ejbName*>:<*endpoint-address-uri*> Specify the endpoint address URI for an EJB Web service in an Application EAR file using this syntax: -wsEndpointAddressURI:<*jarFileInEar*>:<ejbName>:<*endpoint-address-uri*>
wsStyle	Specify the Web service style (DOCUMENT, RPC or WRAPPED) of the generated WSDL binding used when exposing an EJB as a Web service using this syntax: -wsStyle <*style*> Specify the Web service style (DOCUMENT, RPC or WRAPPED) of the generated WSDL binding used when exposing an EJB as a Web service contained in an application EAR file using this syntax: -wsStyle:<*jarFileInEar*>:<*style*>
wsUse	Specify the use (LITERAL or ENCODED) of items in the generated WSDL binding when exposing an EJB as a Web service using this syntax: -wsUse <*use*> Specify the use (LITERAL or ENCODED) of items in the generated WSDL binding when exposing an EJB as a Web service in an application EAR file using this syntax: -wsUse:<*jarFileInEar*>:<*use*>
wsWebAppName	Override the default name for the Web application generated from an EJB Web Service using this syntax: -wsWebAppName <*webappname*> Override default name for the Web application generated from an EJB Web Service in an application EAR file using this syntax: -wsWebAppName:<*jarFileInEar*>:<*webappname*>
entity	The file that you are deploying. *entity* should be located in the current directory, or provide the full path using the contextpath option.
contextpath	Specify the contextPath for Web application deployment. The default is the name of the WAR file. If -package is specified, then the package is the context 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"/>
```

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 109.

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"/>
```

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"/>
```

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.
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" />
```

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.

Option	Description

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.
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.

Option	Description
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.
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."

Option	Description
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).
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"/>
```

Index

E

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X