Artix™

Command Line Reference

Version 4.1, September 2006

Making Software Work Together™

Updated: March 1, 2007

Contents

Preface

What is Covered in this Book

The *Artix Command Line Reference* provides a reference guide to the command line tools provided with Artix.

Who Should Read this Book

The *Artix Command Line Reference* is intended for Artix programmers. This guide assumes that the reader is familiar with the basics of WSDL and XML schemas. A basic knowledge of Artix concepts is presumed.

The Artix Library

The Artix documentation library is organized in the following sections:

- Getting Started
- Designing Artix Solutions
- Configuring and Managing Artix Solutions
- Using Artix Services
- Integrating Artix Solutions
- Integrating with Management Systems
- Reference
- Artix Orchestration

Getting Started

The books in this section provide you with a background for working with Artix. They describe many of the concepts and technologies used by Artix. They include:

- Release Notes contains release-specific information about Artix.

- Installation Guide describes the prerequisites for installing Artix and the procedures for installing Artix on supported systems.
- Getting Started with Artix describes basic Artix and WSDL concepts.
- Using Artix Designer describes how to use Artix Designer to build Artix solutions.
- Artix Technical Use Cases provides a number of step-by-step examples of building common Artix solutions.

Designing Artix Solutions

The books in this section go into greater depth about using Artix to solve real-world problems. They describe how to build service-oriented architectures with Artix and how Artix uses WSDL to define services:

- Building Service-Oriented Infrastructures with Artix provides an overview of service-oriented architectures and describes how they can be implemented using Artix.
- Writing Artix Contracts describes the components of an Artix contract. Special attention is paid to the WSDL extensions used to define Artix-specific payload formats and transports.

Developing Artix Solutions

The books in this section how to use the Artix APIs to build new services:

- Developing Artix Applications in C++ discusses the technical aspects of programming applications using the C++ API.
- Developing Advanced Artix Plug-ins in C++ discusses the technical aspects of implementing advanced plug-ins (for example, interceptors) using the C++ API.
- Developing Artix Applications in Java discusses the technical aspects of programming applications using the Java API.

Configuring and Managing Artix Solutions

This section includes:

- Configuring and Deploying Artix Solutions explains how to set up your Artix environment and how to configure and deploy Artix services.
- Managing Artix Solutions with JMX explains how to monitor and manage an Artix runtime using Java Management Extensions.

Using Artix Services

The books in this section describe how to use the services provided with Artix:

- Artix Router Guide explains how to integrate services using the Artix router.
- Artix Locator Guide explains how clients can find services using the Artix locator.
- Artix Session Manager Guide explains how to manage client sessions using the Artix session manager.
- Artix Transactions Guide, C++ explains how to enable Artix C++ applications to participate in transacted operations.
- Artix Transactions Guide, Java explains how to enable Artix Java applications to participate in transacted operations.
- Artix Security Guide explains how to use the security features in Artix.

Integrating Artix Solutions

The books in this section describe how to integrate Artix solutions with other middleware technologies.

- Artix for CORBA provides information on using Artix in a CORBA environment.
- Artix for J2EE provides information on using Artix to integrate with J2EE applications.

For details on integrating with Microsoft's .NET technology, see the documentation for Artix Connect.

Integrating with Management Systems

The books in this section describe how to integrate Artix solutions with a range of enterprise and SOA management systems. They include:

- IBM Tivoli Integration Guide explains how to integrate Artix with the IBM Tivoli enterprise management system.
- BMC Patrol Integration Guide explains how to integrate Artix with the BMC Patrol enterprise management system.
- CA-WSDM Integration Guide explains how to integrate Artix with the CA-WSDM SOA management system.
- AmberPoint Integration Guide explains how to integrate Artix with the AmberPoint SOA management system.

Reference

These books provide detailed reference information about specific Artix APIs, WSDL extensions, configuration variables, command-line tools, and terms. The reference documentation includes:

- Artix Command Line Reference
- Artix Configuration Reference
- Artix WSDL Extension Reference
- Artix Java API Reference
- Artix C++ API Reference
- Artix .NET API Reference
- Artix Glossary

Artix Orchestration

These books describe the Artix support for Business Process Execution Language (BPEL), which is available as an add-on to Artix. These books include:

- Artix Orchestration Release Notes
- Artix Orchestration Installation Guide
- Understanding Artix Orchestration
- Artix Orchestration Administration Console Help.

Getting the Latest Version

The latest updates to the Artix documentation can be found at http://www.iona.com/support/docs.

Compare the version dates on the web page for your product version with the date printed on the copyright page of the PDF edition of the book you are reading.

Searching the Artix Library

You can search the online documentation by using the **Search** box at the top right of the documentation home page:

http://www.iona.com/support/docs

To search a particular library version, browse to the required index page, and use the **Search** box at the top right, for example:

http://www.iona.com/support/docs/artix/4.0/index.xml

You can also search within a particular book. To search within a HTML version of a book, use the **Search** box at the top left of the page. To search within a PDF version of a book, in Adobe Acrobat, select **Edit**|**Find**, and enter your search text.

Artix Online Help

Artix Designer and Artix Orchestration Designer include comprehensive online help, providing:

- Step-by-step instructions on how to perform important tasks
- A full search feature
- Context-sensitive help for each screen

There are two ways that you can access the online help:

- Select **Help**|**Help Contents** from the menu bar. The help appears in the contents panel of the Eclipse help browser.
- Press **F1** for context-sensitive help.

In addition, there are a number of cheat sheets that guide you through the most important functionality in Artix Designer and Artix Orchestration Designer. To access these, select **Help**|**Cheat Sheets**.

Artix Glossary

The Artix Glossary is a comprehensive reference of Artix terms. It provides quick definitions of the main Artix components and concepts. All terms are defined in the context of the development and deployment of Web services using Artix.

Additional Resources

The IONA Knowledge Base contains helpful articles written by IONA experts about Artix and other products.

The IONA Update Center contains the latest releases and patches for IONA products.

If you need help with this or any other IONA product, go to IONA Online Support.

Comments, corrections, and suggestions on IONA documentation can be sent to docs-support@iona.com .

Document Conventions

Typographical conventions

This book uses the following typographical conventions:

`Fixed width`	Fixed width (courier font) in normal text represents portions of code and literal names of items such as classes, functions, variables, and data structures. For example, text might refer to the `IT_Bus::AnyType` class. Constant width paragraphs represent code examples or information a system displays on the screen. For example: `#include <stdio.h>`
`Fixed width italic`	Fixed width italic words or characters in code and commands represent variable values you must supply, such as arguments to commands or path names for your particular system. For example: `% cd /users/`*`YourUserName`*
Italic	Italic words in normal text represent *emphasis* and introduce *new terms*.
Bold	Bold words in normal text represent graphical user interface components such as menu commands and dialog boxes. For example: the **User Preferences** dialog.

Keying Conventions

This book uses the following keying conventions:

No prompt	When a command's format is the same for multiple platforms, the command prompt is not shown.
`%`	A percent sign represents the UNIX command shell prompt for a command that does not require root privileges.
`#`	A number sign represents the UNIX command shell prompt for a command that requires root privileges.
`>`	The notation > represents the MS-DOS or Windows command prompt.
. . . . . .	Horizontal or vertical ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.
`[]`	Brackets enclose optional items in format and syntax descriptions.
`{}`	Braces enclose a list from which you must choose an item in format and syntax descriptions.
`|`	In format and syntax descriptions, a vertical bar separates items in a list of choices enclosed in `{}` (braces). In graphical user interface descriptions, a vertical bar separates menu commands (for example, select **File|Open**).

CHAPTER 1

Generating WSDL

Artix provides a number of command line tools for generating WSDL.

In this chapter

This chapter discusses the following topics:

Generating from Java Classes

Overview

Artix supplies a command line tool, `javatowsdl`, that generates the logical portion of an Artix contract for existing Java class files. `javatowsdl` uses the mapping rules described in Sun's JAX-RPC 1.1 specification.

JAVATOWSDL

Synopsis

```
javatowsdl [-t namespace][-x namespace][-i porttype][-o
file][-useTypes][-qualified][-v][-h][-L file][q][-verbose]
ClassName
```

Options

The command has the following options:

`-t` *`namespace`*	Specifies the target namespace of the generated WSDL document. By default, the java package name will be used as the target namespace. If no package name is specified, the generated target namespace will be `http:\\www.iona.com\`*`ClassName`*.
`-x` *`namespace`*	Specifies the target namespace of the XMLSchema information generated to represent the data types inside the WSDL document.By default, the generated target namespace of the XMLSchema will be `http:\\www.iona.com\`*`ClassName`*`\xsd`.
`-i` *`porttype`*	Specifies the name of the generated `<portType>` in the WSDL document. By default the name of the class from which the WSDL is generated is used.
`-o` *`file`*	Specifies output file into which the WSDL is written.
`-useTypes`	Specifies that the generated WSDL will use types in the WSDL message parts. By default, messages are generated using wrapped `doc/literal` style. A wrapper element with a sequence will be created to hold method parameters.
`-qualified`	Specifies that the generated WSDL is fully qualified.
`-v`	Displays the tool's version.
`-h`	Displays the tool's usage statement.

`-L` *`file`*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-quiet`	Specifies that the tool runs in quiet mode.
`-verbose`	Specifies that the tool runs in verbose mode.

The generated WSDL will not contain any physical details concerning the payload formats or network transports that will be used when exposing the service. You will need to add this information manually.

Note: When generating contracts, `javatowsdl` will add newly generated WSDL to an existing contract if a contract of the same name exists. It will not generate a new file or warn you that a previous contract exists.

Generating from CORBA IDL

Overview

IONA's IDL compiler supports several command line flags that specify how to create a WSDL file from an IDL file. The default behavior of the tool is to create WSDL file that uses wrapped doc/literal style messages. Wrapped doc/literal style messages have a single part, defined using an element, that wraps all of the elements in the message.

IDLTOWSDL

Synopsis

```
idltowsdl [-useypes][-unwrap][-a address][-f file][-o dir][-s
type][-r file][-L file][-P file][-w namespace][-x namespace][-t
namespace][-T file][-n file][-b][-I
idlDir][-qualified][-inline][-3][-fasttrack][-interface
name][-soapaddr port][-e encoding][-L
file][-quiet][-h][-verbose][-v] idlfile
```

Options

The command has the following options:

`-usetypes`	Generate rpc style messages. rpc style messages have parts defined using XMLSchema types instead of XML elements.
`-unwrap`	Generate unwrapped doc/literal messages. Unwrapped messages have parts that represent individual elements. Unlike wrapped messages, unwrapped messages can have multiple parts and are not allowed by the WS-I.
`-a` *`address`*	Specifies an absolute address through which the object reference may be accessed. The *`address`* may be a relative or absolute path to a file, or a corbaname URL
`-f` *`file`*	Specifies a file containing a string representation of an object reference. The object reference is placed in the `corba:address` element in the `port` definition of the generated service. The *`file`* must exist when you run the IDL compiler.
`-o` *`dir`*	Specifies the directory into which the WSDL file is written.

`-s` *`type`*	Specifies the XMLSchema type used to map the IDL `sequence<octet>` type. Valid values are `base64Binary` and `hexBinary`. The default is `base64Binary`.
`-r` *`file`*	Specify the pathname of the schema file imported to define the `Reference` type. If the `-r` option is not given, the idl compiler gets the schema file pathname from `etc/idl.cfg`.
`-L` *`file`*	Specifies that the logical portion of the generated WSDL specification into is written to *`file`*. *`file`* is then imported into the default generated file.
`-P` *`file`*	Specifies that the physical portion of the generated WSDL specification into is written to *`file`*. *`file`* is then imported into the default generated file.
`-w` *`namespace`*	Specifies the namespace to use for the WSDL `targetNamespace`. The default is `http://schemas.iona.com/idl/`*`idl_name`*.
`-x` *`namespace`*	Specifies the namespace to use for the Schema `targetNamespace`. The default is `http://schemas.iona.com/idltypes/`*`idl_name`*.
`-t` *`namespace`*	Specifies the namespace to use for the CORBA TypeMapping `targetNamespace`. The default is `http://schemas.iona.com/typemap/corba/`*`idl_name`*.
`-T` *`file`*	Specifies that the schema types are to be generated into a separate file. The schema file is included in the generated contract using an import statement. This option cannot be used with the `-n` option.
`-n` *`file`*	Specifies that a schema file, *`file`*, is to be included in the generated contract by an import statement. This option cannot be used with the `-T` option.
`-b`	Specifies that bounded strings are to be treated as unbounded. This eliminates the generation of the special types for the bounded string.
`-I` *`idlDir`*	Specify a directory to be included in the search path for the IDL preprocessor. You can use this flag multiple times.
`-qualified`	Generates fully qualified WSDL.

`-inline`	Generates a contract that includes all imported documents in-line. This overrides all options that specify that a section of the contract is to be imported.
`-3`	Use relaxed IDL grammar checking semantics to allow IDL used by Orbix 3 to be parsed.
`-fasttrack`	Use the fasttrack wizard. You must also use the `-interface` and `-soapaddr` flags with this option. This option also adds a SOAP port and a route between the generated CORBA port and the generated SOAP port.
`-interface` *name*	Specifies the IDL interface for which WSDL will be generated by the fastrack wizard.
`-soapaddr` *port*	Specifies the SOAP address to use in the generated `port` element when using the fasttrack wizard.
`-e` *encoding*	Specifies the value for the generated WSDL document's xml encoding attribute. The default is UTF-8.
`-L` *file*	Specifies the location of your license file. The default is `IT_PRODUCT_DIR\etc\license.txt`.
`-quiet`	Specifies that the tool runs in quiet mode.
`-h`	Displays the tool's usage message.
`-verbose`	Specifies that the tool runs in verbose mode.
`-v`	Displays the tool's version.

Generating from a COBOL Copybook

Overview

Artix provides a command line tool, `colboltowsdl`, that will import COBOL copybook data and generate an Artix contract containing a fixed binding to define the COBOL interface for Artix applications.

COLBOLTOWSDL

Synopsis

```
coboltowsdl -b binding -op operation -im [inmessage:]incopybook [-om
[outmessage:]outcopybook][-fm [faultmessage:]faultbook][-i
portType][-t target][-x
schema_name][-useTypes][-oneway][-qualified][-o file][-L
file][-quiet][-h][-v][-verbose]
```

Parameters

The command has the following required parameters:

`-b binding`	Specifies the name for the generated binding.
`-op operation`	Specifies the name for the generated operation.
`-im [inmessage:]incopybook`	Specifies the name of the input message and the copybook file from which the data defining the message is taken. The input message name, *`inmessage`*, is optional. However, if the copybook has more than one `01` levels, you will be asked to choose the one you want to use as the input message.

Options

The command has the following options:

`-om [outmessage:]outcopybook`	Specifies the name of the output message and the copybook file from which the data defining the message is taken. The output message name, *`outmessage`*, is optional. However, if the copybook has more than one `01` levels, you will be asked to choose the one you want to use as the output message.

`-fm [faultmessage:]faultbook`	Specifies the name of a fault message and the copybook file from which the data defining the message is taken. The fault message name, *`faultmessage`*, is optional. However, if the copybook has more than one `01` levels, you will be asked to choose the one you want to use as the fault message. You can specify more than one fault message.
`-i portType`	Specifies the name of the port type in the generated WSDL. Defaults to *`binding`*`PortType`.[a]
`-t target`	Specifies the target namespace for the generated WSDL. Defaults to `http://www.iona.com/`*`binding`*.
`-x schema_name`	Specifies the namespace for the schema in the generated WSDL. Defaults to `http://www.iona.com/`*`binding`*`/types`.
`-useTypes`	Specifies that the generated WSDL will use `type` elements. Default is to generate `element` elements for schema types.
`-oneway`	Specifies that the operation does not have a response message.
`-qualified`	Specifies that the `schema` element in the generated WSDL has its `elementFormDefault` and `attributeFormDefault` attributes set to `qualified`.
`-o file`	Specifies the name of the generated WSDL file. Defaults to *`binding`*`.wsdl`.
`-L file`	Specifies the location of your Artix license file. The default is `IT_PRODUCT_DIR\etc\license.txt`.
`-quiet`	Specifies that the tool runs in quiet mode. No output will be shown on the console. This includes error messages.
`-h`	Specifies that the tool will display a usage message.
`-v`	Displays the version of the tool.
`-verbose`	Specifies that the tool runs in verbose mode.

a. If *binding* ends in `Binding` or `binding`, it is stripped off before being used in any of the default names.

Once the new contract is generated, you will still need to add the port information before you can use the contract to develop an Artix solution.

Generating from an XMLSchema Document

Overview

Artix provides a command line tool, `xsdtowsdl`, that will import an XMLSchema document and generate an Artix contract containing a `types` element populated by the types defined in the XMLSchema document. The rest of the contract will be empty.

XDSTOWSDL

Synopsis

```
xsdtowsdl [-t namespace][-n name][-d dir][-o
file][-?][-v][-verbose][-L file][-quiet][-h][-verbose][-v] xsdurl
```

Options

The command has the following options:

`-t` *`namespace`*	Specifies the target namespace for the generated contract. The default is to use the Artix target namespace.
`-n` *`name`*	Specifies the name for the generated contract and is the value of the `name` attribute in the contract's root `definitions` element. The default is to use the schema document's file name.
`-d` *`dir`*	Specifies the output directory for the generated contract.
`-o` *`file`*	Specifies the filename for the generated contract. Defaults to the filename of the imported schema document. For example, if the imported schema document is stored in `maxwell.xsd` the resulting contract will be `maxwell.wsdl`.
`-L` *`file`*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-quiet`	Specifies that the tool runs in quiet mode.
`-h`	Displays the tool's usage message.
`-verbose`	Specifies that the tool runs in verbose mode.

`-v`	Displays the tool's version.

Generating from an Artix Database Configuration Document

Overview

Artix provides a command line tool, `dbconfigtowsdl`, that will import an Artix database configuration document and generate an Artix contract defining a service that represents the database operations defined in the document.

DBCONFIGTOWSDL

Synopsis

```
dbconfigtowsdl [-t bindingAddress][-fasttrack][-plugin][-d
dir][-source dir][-h][-v][-verbose][-quiet] dbconfigurl
```

Options

The command has the following options:

`-t` *`bindingAddress`*	Specifies the address to use in the `port` element of the generated WSDL. This flag is only valid when `-fasttrack` is also used. The default is `http://localhost:9000/DBConnection`.
`-fasttrack`	Specifies that the tool will generate a default SOAP binding and HTTP endpoint for the database operations. In addition, the tool will generate the code for the intermediary required to expose the operations as a service.
`-plugin`	Specifies that the intermediary is generated as an Artix plug-in. This flag is only valid when `-fastttrack` is also used.
`-d` *`dir`*	Specifies the output directory for the generated WSDL file. The default is the local directory. When `-fasttrack` is used, the default is `etc`.
`-source` *`dir`*	Specifies the output directory for the generated code. This flag is only valid when `-fasttrack` is also used. The default is `java`.

`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.
`-verbose`	Specifies that the tool runs in verbose mode.
`-quiet`	Specifies that the tool runs in quiet mode.

CHAPTER 2

Adding Bindings

Artix provides a tools for adding bindings to WSDL.

In this chapter

This chapter discusses the following topics:

Adding a SOAP Binding

Overview

Artix provides a tool, `wsdltosoap`, that will generate a SOAP binding from an existing logical interface defined in a WSDL `<portType>`. The tool will generate a new contract which includes the generated SOAP binding.

WSDLTOSOAP

Synopsis

```
wsdltosoap -i portType -n namespace wsdl_file [-soapversion
{1.1|1.2}][-b binding][-d dir][-o file][-style {document|rpc}][-use
{literal|encoded}][-L file][-quiet][-h][-verbose][-v]
```

Parameters

The command has the following required parameters:

`-i` *`portType`*	Specifies the name of the port type being mapped to a SOAP binding.
`-n` *`namespace`*	Specifies the namespace to use for the SOAP binding.
`wsdl_file`	Specifies the WSDL file in which the logical binding is defined.

Options

The command has the following options:

`-soapversion {1.1|1.2}`	Specifies the SOAP version of the generated binding. Defaults to `1.1`.
`-b` *`binding`*	Specifies the name for the generated SOAP binding. Defaults to *`portTypeBinding`*.
`-d` *`dir`*	Specifies the directory into which the new WSDL file is written.
`-o file`	Specifies the name of the generated WSDL file. Defaults to *`wsdl_file`*`-soap.wsdl`.
`-style`	Specifies the encoding style to use in the SOAP binding. Defaults to `document`.
`-use`	Specifies how the data is encoded. Default is `literal`.
`-L` *`file`*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.

`-quiet`	Specifies that the tool runs in quiet mode.
`-h`	Displays the tool's usage message.
`-v`	Displays the tool's version.
`-verbose`	Specifies that the tool runs in verbose mode.

Notes

`wsdltosoap` does not support the the generatoin of `document`/`encoded` SOAP bindings.

Adding a CORBA Binding

Overview

The `wsdltocorba` tool adds CORBA binding information to an existing Artix contract. The generated WSDL file will also contain a CORBA port with no address specified.

WSDLTOCORBA

Synopsis

```
wsdltocorba -corba -i portType [-d dir][-b binding][-o file][-props
namespace][-wrapped][-L file][-quiet][-verbose][-h][-v] wsdl_file
```

Parameters

The command has the following required parameters:

`-corba`	Instructs the tool to generate a CORBA binding for the specified port type.
`-i portType`	Specifies the name of the port type being mapped to a CORBA binding.
`wsdl_file`	Specifies the name of the WSDL file containing the logical interface to which the CORBA binding is mapped.

Options

The command has the following options:

`-d dir`	Specifies the directory into which the new WSDL file is written.
`-b binding`	Specifies the name for the generated CORBA binding. Defaults to *`portType`*`Binding`.
`-o file`	Specifies the name of the generated WSDL file. Defaults to *`wsdl_file`*`-corba.wsdl`.
`-props namespace`	Specifies the namespace to use for the generated CORBA typemap
`-wrapped`	Specifies that the generated CORBA binding uses wrapper types.
`-L file`	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-quiet`	Specifies that the tool runs in quiet mode.
`-verbose`	Specifies that the tool runs in verbose mode.

`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.

Notes

By combining the `-idl` and `-corba` flags with `wsdltocorba`, you can generate a CORBA binding for a logical operation and then generate the IDL for the generated CORBA binding. When doing so, you must also use the `-i` *`portType`* flag to specify the port type from which to generate the binding and the `-b` *`binding`* flag to specify the name of the binding to from which to generate the IDL.

CHAPTER 3

Adding Endpoints

Artix provides a tools for adding endpoint definitions to WSDL.

In this chapter

This chapter discusses the following topics:

Adding an HTTP Endpoint

Overview

The Artix `wsdltoservice` tool can generate an HTTP endpoint from an existing logical interface defined in a WSDL `portType` element.

WSDLTOSERVICE -transport http/soap

Synopsis

```
wsdltoservice -transport soap/http [-e service] [-t port] [-b binding]
[-a address] [-hssdt serverSendTimeout] [-hscvt
serverReceiveTimeout] [-hstrc trustedRootCertificates] [-hsuss
useSecureSockets] [-hsct contentType] [-hscc
serverCacheControl] [-hsscse supressClientSendErrors] [-hsscre
supressClientReceiveErrors] [-hshka honorKeepAlive] [-hsmps
serverMultiplexPoolSize] [-hsrurl redirectURL] [-hscl
contentLocation] [-hsce contentEncoding] [-hsst serverType] [-hssc
serverCentificate] [-hsscc serverCentificateChain] [-hsspk
serverPrivateKey] [-hsspkp serverPrivateKeyPassword] [-hcst
clientSendTimeout] [-hccvt clientReceiveTimeout] [-hctrc
trustedRootCertificates] [-hcuss useSecureSockets] [-hcct
contentType] [-hccc clientCacheControl] [-hcar autoRedirect] [-hcun
userName] [-hcp password] [-hcat clientAuthorizationType] [-hca
clientAuthorization] [-hca accept] [-hcal acceptLanguage] [-hcae
acceptEncoding] [-hch host] [-hccn clientConnection] [-hcck
cookie] [-hcbt browserType] [-hcr referer] [-hcps proxyServer] [-hcpun
proxyUserName] [-hcpp proxyPassword] [-hcpat proxyAuthorizationType]
[-hcpa proxyAuthorization] [-hccce clientCertificate] [-hcccc
clientCertificateChain] [-hcpk clientPrivateKey] [-hcpkp
clientPrivateKeyPassword] [-o file] [-d dir] [-L
file] [-quiet] [-verbose] [-h] [-v] wsdlurl
```

Options

The command has the following options:

`-transport soap/http`	If the payload being sent over the wire is SOAP, use `-transport soap`. For all other payloads use `-transport http`.
`-e service`	Specifies the name of the generated service.
`-t port`	Specifies the value of the `name` attribute of the generated `port` element.
`-b binding`	Specifies the name of the binding for which the service is generated.

`-a address`	Specifies the value used in the `address` element of the port.
`-hssdt serverSendTimeout`	Specifies the number if milliseconds that the server can continue to try to send a response to the client before the connection is timed out.
`-hscvt serverReceiveTimeout`	Specifies the number of milliseconds that the server can continue to try to receive a request from the client before the connection is timed out.
`-hstrc trustedRootCertificates`	Specifies the full path to the X509 certificate for the certificate authority.
`-hsuss useSecureSockets`	Specifies if the server uses secure sockets. Valid values are `true` or `false`.
`-hsct contentType`	Specifies the media type of the information being sent in a server response.
`-hscc serverCacheControl`	Specifies directives about the behavior that must be adhered to by caches involved in the chain comprising a request from a client to a server.
`-hsscse supressClientSendErrors`	Specifies whether exceptions are thrown when an error is encountered on receiving a client request. Valid values are `true` or `false`.
`-hsscre supressClientReceiveErrors`	Specifies whether exceptions are thrown when an error is encountered on sending a response to a client. Valid values are `true` or `false`.
`-hshka honorKeepAlive`	Specifies if the server honors client keep-alive requests. Valid values are `true` or `false`.
`-hsmps serverMultiplexPoolSize`	
`-hsrurl redirectURL`	Specifies the URL to which the client request should be redirected if the URL specified in the client request is no longer appropriate for the requested resource.
`-hscl contentLocation`	Specifies the URL where the resource being sent in a server response is located.

`-hsce` *contentEncoding*	Specifies what additional content codings have been applied to the information being sent by the server, and what decoding mechanisms the client therefore needs to retrieve the information.
`-hsst` *serverType*	Specifies what type of server is sending the response to the client.
`-hssc` *serverCentificate*	Specifies the full path to the X509 certificate issued by the certificate authority for the server.
`-hsscc` *serverCentificateChain*	Specifies the full path to the file that contains all the certificates in the chain.
`-hsspk` *serverPrivateKey*	Specifies the full path to the private key that corresponds to the X509 certificate specified by *serverCertificate*.
`-hsspkp` *serverPrivateKeyPassword*	Specifies a password that is used to decrypt the private key.
`-hcst` *clientSendTimeout*	Specifies the number of milliseconds that the client can continue to try to send a request to the server before the connection is timed out.
`-hccvt` *clientReceiveTimeout*	Specifies the number of milliseconds that the client can continue to try to receive a response from the server before the connection is timed out.
`-hctrc` *trustedRootCertificates*	Specifies the full path to the X509 certificate for the certificate authority.
`-hcuss` *ueSecureSockets*	Specifies if the client uses secure sockets. Valid values are *true* or *false*.
`-hcct` *contentType*	Specifies the media type of the data being sent in the body of the client request.
`-hccc` *clientCacheControl*	Specifies directives about the behavior that must be adhered to by caches involved in the chain comprising a request from a client to a server.
`-hcar` *autoRedirect*	Specifies if the server should automatically redirect client requests.
`-hcun` *userName*	Specifies the username the client uses to register with servers.

`-hcp` *password*	Specifies the password the client uses to register with servers.
`-hcat` *clientAuthorizationType*	Specifies the authorization mechanisms the client uses when contacting servers.
`-hca` *clientAuthorization*	Specifies the authorization credentials used to perform the authorization.
`-hca` *accept*	Specifies what media types the client is prepared to handle.
`-hcal` *acceptLanguage*	Specifies what language the client prefers for the purposes of receiving a response
`-hcae` *acceptEncoding*	Specifies what content codings the client is prepared to handle.
`-hch` *host*	Specifies the internet host and port number of the resource on which the client request is being invoked.
`-hccn` *clientConnection*	Specifies if the client will open a new connection for each request or if it will keep the original one open. Valid values are `close` and `Keep-Alive`.
`-hcck` *cookie*	Specifies a static cookie to be sent to the server.
`-hcbt` *browserType*	Specifies information about the browser from which the client request originates.
`-hcr` *referer*	Specifies the value for the client's referring entity.
`-hcps` *proxyServer*	Specifies the URL of the proxy server, if one exists along the message path.
`-hcpun` *proxyUserName*	Specifies the username that the client uses to authorize with proxy servers.
`-hcpp` *proxyPassword*	Specifies the password that the client uses to authorize with proxy servers.
`-hcpat` *proxyAuthorizationType*	Specifies the authorization mechanism the client uses with proxy servers.
`-hcpa` *proxyAuthorization*	Specifies the actual data that the proxy server should use to authenticate the client.

`-hccce` *clientCertificate*	Specifies the full path to the X509 certificate issued by the certificate authority for the client.
`-hcccc` *clientCertificateChain*	Specifies the full path to the file that contains all the certificates in the chain.
`-hcpk` *clientPrivateKey*	Specifies the full path to the private key that corresponds to the X509 certificate specified by *clientCertificate*.
`-hcpkp` *clientPrivateKeyPassword*	Specifies a password that is used to decrypt the private key.
`-o` *file*	Specifies the filename for the generated contract. The default is to append `-service` to the name of the imported contract.
`-d` *dir*	Specifies the output directory for the generated contract.
`-L` *file*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-quiet`	Specifies that the tool runs in quiet mode.
`-verbose`	Specifies that the tool runs in verbose mode.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.

Adding a CORBA Endpoint

Overview

The Artix `wsdltoservice` tool can generate a CORBA endpoint from an existing logical interface defined in a WSDL `portType` element.

WSDLTOSERVICE -transport corba

Synopsis

```
wsdltoservice -transport corba [-e service][-t port][-b binding][-a
address][-poa poaName][-sid serviceId][-pst persists][-o file][-d
dir][-L file][-quiet][-verbose][-h][-v] wsdlurl
```

Options

The command has the following options:

`-e` *service*	Specifies the name of the generated CORBA service.
`-t` *port*	Specifies the value of the `name` attribute of the generated `port` element.
`-b` *binding*	Specifies the name of the binding for which the service is generated.
`-a` *address*	Specifies the value used in the `corba:address` element of the port.
`-poa` *poaName*	Specifies the value of the POA name policy.
`-sid` *serviceId*	Specifies the value of the ID assignment policy.
`-pst` *persists*	Specifies the value of the persistence policy. Valid values are `true` and `false`.
`-o` *file*	Specifies the filename for the generated contract. The default is to append `-service` to the name of the imported contract.
`-d` *dir*	Specifies the output directory for the generated contract.
`-L` *file*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-quiet`	Specifies that the tool runs in quiet mode.
`-verbose`	Specifies that the tool runs in verbose mode.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.

Adding an IIOP Endpoint

Overview

The Artix `wsdltoservice` tool can generate an IIOP endpoint from an existing logical interface defined in a WSDL `portType` element.

WSDLTOSERVICE -transport iiop

Synopsis

```
wsdltoservice -transport iiop [-e service][-t port][-b binding][-a
address][-poa poaName][-sid serviceId][-pst persists][-paytype
payload][-o file][-d dir][-L file][-quiet][-verbose][-h][-v]
wsdlurl
```

Options

The command has the following options:

`-e` *`service`*	Specifies the name of the generated IIOP service.
`-t` *`port`*	Specifies the value of the `name` attribute of the generated `port` element.
`-b` *`binding`*	Specifies the name of the binding for which the service is generated.
`-a` *`address`*	Specifies the value used in the `<iiop:address>` element of the port.
`-poa` *`poaName`*	Specifies the value of the POA name policy.
`-sid` *`serviceId`*	Specifies the value of the ID assignment policy.
`-pst` *`persists`*	Specifies the value of the persistence policy. Valid values are `true` and `false`.
`-paytype` *`payload`*	Specifies the type of data being sent in the message payloads. Valid values are `string`, `octets`, `imsraw`, `imsraw_binary`, `cicsraw`, and `cicsraw_binary`.
`-o` *`file`*	Specifies the filename for the generated contract. The default is to append `-service` to the name of the imported contract.
`-d` *`dir`*	Specifies the output directory for the generated contract.
`-L` *`file`*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-quiet`	Specifies that the tool runs in quiet mode.

`-verbose`	Specifies that the tool runs in verbose mode.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.

Adding a WebSphere MQ Endpoint

Overview

The Artix `wsdltoservice` tool can generate a WebSphere MQ endpoint from an existing logical interface defined in a WSDL `portType` element.

WSDLTOSERVICE -transport mq

Synopsis

```
wsdltoservice -transport mq [-e service][-t port][-b binding][-sqm
queueManager][-sqn queue][-srqm queueManager][-srqn queue][-smqn
modelQueue][-sus usageStyle][-scs correlationStyle][-sam
accessMode][-sto timeout][-sme expiry][-smp priority][-smi
messageId][-sci correlationId][-sd delivery][-st
transactional][-sro reportOption][-sf format][-sad
applicationData][-sat accountingToken][-scn connectionName][-sc
convert][-scr reusable][-scfp fastPath][-said idData][-saod
originData][-cqm queueManager][-cqn queue][-crqm
queueManager][-crqn queue][-cmqn modelQueue][-cus usageStyle][-ccs
correlationStyle][-cam accessMode][-cto timeout][-cme expiry][-cmp
priority][-cmi messageId][-cci correlationId][-cd delivery][-ct
transactional][-cro reportOption][-cf format][-cad
applicationData][-cat accountingToken][-ccn connectionName][-cc
convert][-ccr reusable][-ccfp fastPath][-caid idData][-caod
originData][-caqn queue][-cui userId][-o file][-d dir][-L
file][-quiet][-verbose][-h][-v] wsdlurl
```

Options

The command has the following options:

`-e service`	Specifies the name of the generated service.
`-t port`	Specifies the value of the `name` attribute of the generated `port` element.
`-b binding`	Specifies the name of the binding for which the service is generated.
`-sqm queueManager`	Specifies the name of the server's queue manager.
`-sqn queue`	Specifies the name of the server's request queue.
`-srqm queueManager`	Specifies the name of the server's reply queue manager.
`-srqn queue`	Specifies the name of the server's reply queue.

`-smqn` *`modelQueue`*	Specifies the name of the server's model queue.
`-sus` *`usageStyle`*	Specifies the value of the server's `UsageStyle` attribute. Valid values are `Peer`, `Requester`, or `Responder`.
`-scs` *`correlationStyle`*	Specifies the value of the server's `CorrelationStyle` attribute. Valid values are `messageId`, `correlationId`, or `messageId copy`.
`-sam` *`accessMode`*	Specifies the value of the server's `AccessMode` attribute. Valid values are `peek`, `send`, `receive`, `receive exclusive`, or `receive shared`.
`-sto` *`timeout`*	Specifies the value of the server's `Timeout` attribute.
`-sme` *`expiry`*	Specifies the value of the server's `MessageExpiry` attribute.
`-smp` *`priority`*	Specifies the value of the server's `MessagePriority` attribute.
`-smi` *`messageId`*	Specifies the value of the server's `MessageId` attribute.
`-sci` *`correlationId`*	Specifies the value of the server's `CorrelationId` attribute.
`-sd` *`delivery`*	Specifies the value of the server's `Delivery` attribute.
`-st` *`transactional`*	Specifies the value of the server's `Transactional` attribute. Valid values are `none`, `internal`, or `xa`.
`-sro` *`reportOption`*	Specifies the value of the server's `ReportOption` attribute. Valid values are `none`, `coa`, `cod`, `exception`, `expiration`, or `discard`.
`-sf` *`format`*	Specifies the value of the server's `Format` attribute.
`-sad` *`applicationData`*	Specifies the value of the server's `ApplicationData` attribute.
`-sat` *`accountingToken`*	Specifies the value of the server's `AccountingToken` attribute.
`-scn` *`connectionName`*	Specifies the name of the connection by which the adapter connects to the queue.

`-sc` *convert*	Specifies if the messages in the queue need to be converted to the system's native encoding. Valid values are `true` or `false`.
`-scr` *reusable*	Specifies the value of the server's `ConnectionReusable` attribute. Valid values are `true` or `false`.
`-scfp` *fastPath*	Specifies the value of the server's `ConnectionFastPath` attribute. Valid values are `true` or `false`.
`-said` *idData*	Specifies the value of the server's `ApplicationIdData` attribute.
`-saod` *originData*	Specifies the value of the server's `ApplicationOriginData` attribute.
`-cqm` *queueManager*	Specifies the name of the client's queue manager.
`-cqn` *queue*	Specifies the name of the client's request queue.
`-crqm` *queueManager*	Specifies the name of the client's reply queue manager.
`-crqn` *queue*	Specifies the name of the client's reply queue.
`-cmqn` *modelQueue*	Specifies the name of the client's model queue.
`-cus` *usageStyle*	Specifies the value of the client's `UsageStyle` attribute. Valid values are `Peer`, `Requester`, or `Responder`.
`-ccs` *correlationStyle*	Specifies the value of the client's `CorrelationStyle` attribute. Valid values are `messageId`, `correlationId`, or `messageId copy`.
`-cam` *accessMode*	Specifies the value of the client's `AccessMode` attribute. Valid values are `peek`, `send`, `receive`, `receive exclusive`, or `receive shared`.
`-cto` *timeout*	Specifies the value of the client's `Timeout` attribute.
`-cme` *expiry*	Specifies the value of the client's `MessageExpiry` attribute.
`-cmp` *priority*	Specifies the value of the client's `MessagePriority` attribute.
`-cmi` *messageId*	Specifies the value of the client's `MessageId` attribute.

`-cci` *correlationId*	Specifies the value of the client's `CorrelationId` attribute.
`-cd` *delivery*	Specifies the value of the client's `Delivery` attribute.
`-ct` *transactional*	Specifies the value of the client's `Transactional` attribute. Valid values are `none`, `internal`, or `xa`.
`-cro` *reportOption*	Specifies the value of the client's `ReportOption` attribute. Valid values are `none`, `coa`, `cod`, `exception`, `expiration`, or `discard`.
`-cf` *format*	Specifies the value of the client's `Format` attribute.
`-cad` *applicationData*	Specifies the value of the client's `ApplicationData` attribute.
`-cat` *accountingToken*	Specifies the value of the client's `AccountingToken` attribute.
`-ccn` *connectionName*	Specifies the name of the connection by which the adapter connects to the queue.
`-cc` *convert*	Specifies if the messages in the queue need to be converted to the system's native encoding. Valid values are `true` or `false`.
`-ccr` *reusable*	Specifies the value of the client's `ConnectionReusable` attribute. Valid values are `true` or `false`.
`-ccfp` *fastPath*	Specifies the value of the client's `ConnectionFastPath` attribute. Valid values are `true` or `false`.
`-caid` *idData*	Specifies the value of the client's `ApplicationIdData` attribute.
`-caod` *originData*	Specifies the value of the client's `ApplicationOriginData` attribute.
`-caqn` *queue*	Specifies the remote queue to which a server will put replies if its queue manager is not on the same host as the client's local queue manager.
`-cui` *userId*	Specifies the value of the client's `UserIdentification` attribute.
`-o` *file*	Specifies the filename for the generated contract. The default is to append `-service` to the name of the imported contract.

`-L` *`file`*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-quiet`	Specifies that the tool runs in quiet mode.
`-verbose`	Specifies that the tool runs in verbose mode.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.
`-d` *`dir`*	Specifies the output directory for the generated contract.

Adding a JMS Endpoint

Overview

The Artix `wsdltoservice` tool can generate a JMS endpoint from an existing logical interface defined in a WSDL `portType` element.

WSDLTOSERVICE -transport jms

Synopsis

```
wsdltoservice -transport jms [-e service][-t port][-b binding][-o
file][-d dir][-jnp propName:propVal]*[-jds (queue/topic)][-jnf
connectionFactoryName][-jdn destinationName][-jrdn
replyDesinationName][-jcun username][-jcp password][-jmt
(text/binary)][-jms messageSelector][-jumi (true/false)][-jtr
(true/false)][-jdsn durableSubscriber][-L
file][-quiet][-verbose][-h][-v] wsdlurl
```

Options

The command has the following options:

`-e service`	Specifies the name of the generated `service` element.
`-t port`	Specifies the value of the `name` attribute of the generated `port` element.
`-b binding`	Specifies the name of the binding for which the service is generated.
`-o file`	Specifies the filename for the generated contract. The default is to append `-service` to the name of the imported contract.
`-d dir`	Specifies the output directory for the generated contract.
`-jnp propName:propVal`	Specifies any optional Java properties to use in connecting to the JNDI provider. This information is used to populate a `JMSNamingProperty` element. You can use this flag multiple times.
`-jds (queue/topic)`	Specifies if the JMS destination is a JMS queue or a JMS topic.

`-jfn` *`connectionFactoryName`*	Specifies the JNDI name bound to the JMS connection factory to use when connecting to the JMS destination.
`-jdn` *`destinationName`*	Specifies the JNDI name of the JMS destination to which Artix connects.
`-jrdn` *`replyDestinationName`*	Specifies the JNDI name of the JMS destination used for replies.
`-jcun` *`username`*	Specifies the username used to connect to the JMS broker.
`-jcp` *`password`*	Specifies the password used to connect to the JMS broker.
`-jmt (text/binary)`	Specifies how the message data will be packaged as a JMS message.
`-jms` *`messageSelector`*	Specifies a message selector to use when pulling messages from the JMS destination.
`-jumi (true/false)`	Specifies if the JMS message id should be used as the correlation id.
`-jtr (true/false)`	Specifies if the services uses local JMS transactions when processing requests.
`-jdsn` *`durableSubscriber`*	Specifies the name of the durable subscription to use.
`-L` *`file`*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-quiet`	Specifies that the tool runs in quiet mode.
`-verbose`	Specifies that the tool runs in verbose mode.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.

Adding a Tibco Endpoint

Overview

The Artix `wsdltoservice` tool can generate a Tibco endpoint from an existing logical interface defined in a WSDL `portType` element.

WSDLTOSERVICE -transport tibrv

Synopsis

```
wsdltoservice -transport tibrv [-e service][-t port][-b
binding][-tss subject][-tcst subject][-tbt bindingType][-tcl
callbackLevel][-trdt timeout][-tts transportService][-ttn
transportNetwork][-ttbm batchMode][-tqp priority][-tqlp
queueLimitPolicy][-tqme queueMaxEvents][-tqda
queueDiscardAmount][-tcs cmSupport][-tctsn
cmTransportServerName][-tctcn cmTransportClientName][-tctro
cmTransportRequestOld][-tctln cmTransportLedgerName][-tctsl
cmTransportSyncLedger][-tctra cmTransportRelayAgent][-tctdtl
cmTransportDefaultTimeLimit][-tclca
cmListenerCancelAgreements][-tcqtsn
cmQueueTransportServerName][-tcqtcn
cmQueueTransportClientName][-tcqtww
cmQueueTransportWorkerWeight][-tcqtws
cmQueueTransportWorkerTasks][-tcqtsw
cmQueueTransportSchedulerWeight][-tcqtsh
cmQueueTransportSchedulerHeartbeat][-tcqtsa
cmQueueTransportSchedulerActivation][-tcqtct
cmQueueTransportCompleteTime][-tmnfv messageNameFieldValue][-tmnfp
messageNameFieldPath][-tbfi bindingFieldId][-tbfn
bindingFieldName][-o file][-d dir][-L
file][-quiet][verbose][-h][-v] wsdlurl
```

Options

The command has the following options:

`-e service`	Specifies the name of the generated service.
`-t port`	Specifies the value of the `name` attribute of the generated `port` element.
`-b binding`	Specifies the name of the binding for which the service is generated.
`-tss subject`	Specifies the subject to which the server listens.

`-tbt` *`bindingType`*	Specifies the message binding type. Valid vales are `msg`, `xml`, `opaque`, or `string`.
`-tcl` *`callbackLevel`*	Specifies the server-side callback level when TIB/RV system advisory messages are received. Valid values are `INFO`, `WARN`, or `ERROR`.
`-trdt` *`timeout`*	Specifies the client-side response receive dispatch time-out.
`-tts` *`transportService`*	Specifies the UDP service name or port for TibrvNetTransport.
`-ttn` *`transportNetwork`*	Specifies the binding network addresses for TibrvNetTransport.
`-ttbm` *`batchMode`*	Specifies if the TIB/RV transport uses batch mode to send messages. Valid values are `DEFAULT_BATCH` and `TIMER_BATCH`.
`-tqp` *`priority`*	Specifies the queue priority.
`-tqlp` *`queueLimitPolicy`*	Valid values are `DISCARD_NONE`, `DISCARD_NEW`, `DISCARD_FIRST`, or `DISCARD_LAST`.
`-tqme` *`queueMaxEvents`*	Specifies the queue max events.
`-tqda` *`queueDiscardAmount`*	Specifies the queue discard amount.
`-tcs` *`cmSupport`*	Specifies if Certified Message Delivery support is enabled. Valid values are `true` or `false`.
`-tctsn` *`cmTransportServerName`*	Specifies the server's TibrvCmTransport correspondent name.
`-tctcn` *`cmTransportClientName`*	Specifies the client TibrvCmTransport correspondent name.
`-tctro` *`cmTransportRequestOld`*	Specifies if the endpoint can request old messages on start-up. Valid values are `true` or `false`.
`-tctln` *`cmTransportLedgerName`*	Specifies the TibrvCmTransport ledger file.
`-tctsl` *`cmTransportSyncLedger`*	Specifies if the endpoint uses a synchronous ledger. Valid values are `true` or `false`.

`-tctra` *cmTransportRelayAgent*	Specifies the endpoint's TibrvCmTransport relay agent.
`-tctdtl` *cmTransportDefaultTimeLimit*	Specifies the default time limit for a Certified Message to be delivered.
`-tclca` *cmListenerCancelAgreements*	Specifies if Certified Message agreements are canceled when the endpoint disconnects. Valid values are `true` or `false`.
`-tcqtsn` *cmQueueTransportServerName*	Specifies the server's TibrvCmQueueTransport correspondent name.
`-tcqtcn` *cmQueueTransportClientName*	Specifies the client's TibrvCmQueueTransport correspondent name.
`-tcqtww` *cmQueueTransportWorkerWeight*	Specifies the endpoint's TibrvCmQueueTransport `worker weight`.
`-tcqtws` *cmQueueTransportWorkerTasks*	Specifies the endpoint's TibrvCmQueueTransport `worker tasks` parameter.
`-tcqtsw` *cmQueueTransportSchedulerWeight*	Specifies the TibrvCmQueueTransport `scheduler weight` parameter.
`-tcqtsh` *cmQueueTransportSchedulerHeartbeat*	Specifies the endpoint's TibrvCmQueueTransport `scheduler heartbeat` parameter.
`-tcqtsa` *cmQueueTransportSchedulerActivation*	Specifies the TibrvCmQueueTransport `scheduler activation` parameter.
`-tcqtct` *cmQueueTransportCompleteTime*	Specifies the TibrvCmQueueTransport `complete time` parameter.
`-tmnfv` *messageNameFieldValue*	Specifies the message name field value.
`-tmnfp` *messageNameFieldPath*	Specifies the message name field path.
`-tbfi` *bindingFieldId*	Specifies the binding field id.
`-tbfn` *bindingFieldName*	Specifies the binding field name.

`-o` *`file`*	Specifies the filename for the generated contract. The default is to append `-service` to the name of the imported contract.
`-d` *`dir`*	Specifies the output directory for the generated contract.
`-L` *`file`*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-quiet`	Specifies that the tool runs in quiet mode.
`-verbose`	Specifies that the tool runs in verbose mode.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.

Adding a Tuxedo Service

Overview

The Artix `wsdltoservice` tool can generate a Tuxedo service from an existing logical interface defined in a WSDL `portType` element.

WSDLTOSERVICE -transport tuxedo

Synopsis

```
wsdltoservice -transport tuxedo [-e service][-t port][-b
binding][-tsn tuxService][-tfn tuxService:tuxFunction][-ton
tuxService:operation][-o file][-d dir][-L
file][-quiet][-verbose][-h][-v] wsdlurl
```

Options

The command has the following options:

`-e` *service*	Specifies the name of the generated service.
`-t` *port*	Specifies the value of the `name` attribute of the generated `port` element.
`-b` *binding*	Specifies the name of the binding for which the service is generated.
`-tsn` *tuxService*	Specifies the name the service uses to register with the Tuxedo bulletin board.
`-tfn` *tuxService*`:`*tuxFunction*	Specifies the name of the function to be used on the specified Tuxedo bulletin board.
`-ton` *tuxService*`:`*operation*	Specifies the WSDL operation that is handled by the specified Tuxedo endpoint.
`-o` *file*	Specifies the filename for the generated contract. The default is to append `-service` to the name of the imported contract.
`-d` *dir*	Specifies the output directory for the generated contract.
`-L` *file*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.

`-quiet`	Specifies that the tool runs in quiet mode.
`-verbose`	Specifies that the tool runs in verbose mode.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.

Adding Routes

You can add routes to your Artix contracts from the command line.

Overview

Artix includes a command line tool, `wsdltorouting`, for adding routes to Artix contracts.

WSDLTOROUTING

Synopsis

```
wsdltorouting [-rn name][-ssn service][-spn port][-dsn
service][-dpn port][-on operation][-ta attribute] [-d dir][-o
file][-L file][-quiet][-verbose][-h][-v] wsdlurl
```

Options

You can supply the following optional parameters:

`-rn` *`name`*	Specifies the name of the generated route. If no name is given a unique name will be generated for the route.
`-ssn` *`service`*	Specifies the name of the service to use as the source of the route.
`-spn` *`port`*	Specifies the name of the port to use as the source of the route. The port must correspond to a `port` element in the specified service.
`-dsn` *`service`*	Specifies the name of the service to use as the destination of the route.
`-dpn` *`port`*	Specifies the name of the port to use as the destination of the route. The port must correspond to a `port` element in the specified service.

`-on` *`operation`*	Specifies the name of the operation to use for the route. If the route is port-based, you do not need to use this flag.
`-ta` *`attribute`*	Specifies a transport attribute to use in defining the route. For details on how to specify the transport attributes.
`-d` *`dir`*	Specifies the output directory for the generated contract.
`-o` *`file`*	Specifies the filename of the generated contract.
`-L` *`file`*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-quiet`	Specifies that the tool runs in quiet mode.
`-verbose`	Specifies that the tool runs in verbose mode.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.

CHAPTER 5

Validating WSDL

Artix can validate your contracts to see if they are well-formed WSDL documents. In addition, Artix can validate your contract against the WS-I Basic Profile.

Overview

Artix includes a command line tool, `schemavalidator`, for validating Artix contracts.

SCHEMAVALIDATOR

Synopsis

```
schemavalidator [-d schema-directory]* [-s schema-url]* [-w
WSDL_XSD_URL][-deep][-wsi][-wh wsi-test-tools.home][-tad
BasicProfileAssertions][-L file][-quiet][-verbose][-h][-v]
```

Parameters

You must specify the location of a WSDL contract file, *`WSDL_XSD_URL`*, for the schema validator to work.

Options

You can supply the following optional parameters:

`-d schema-directory`	Specifies the directory used to search for schemas. This switch can appear multiple times.
`-s schema-url`	Specifies the URL of a user specific schema to be included in the validation of the contract. This switch can appear multiple times.

`-deep`	Specifies that the validator is to check all WSDL imports and all WSDL semantics. When using this switch, the tool will also validate the imported WSDL.
`-wsi`	Specifies that the tool is to use the wsi-test-tools from wsi.org to validate the contract.
`-wh` *`wsi-test-tools.home`*	Specifies the base directory of wsi-test-tools.
`-tad` *`BasicProfileAssertions`*	Specifies the URL of the of `BasicProfileTestAssertions.xml` used in wsi-test-tools.
`-L` *`file`*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-quiet`	Specifies that the tool runs in quiet mode.
`-verbose`	Specifies that the tool runs in verbose mode.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.

Transforming XML

Artix includes a command line driven XSLT processor for transforming XML documents.

Overview

Artix includes a command line tool, `xslttransform`, for transforming XML documents.

XSLTTRANSFORM

Synopsis

`xslttransform -IN` *`inputXMLURL`* `-OUT` *`outputXMLURL`* `-XS` *`XSLTURL`* `[-PARAM` *`name value`*`]*`

Parameters

The command has the following parameters:

`-IN` *`inputXMLURL`*	Specifies the URL of the source XML document.
`-OUT` *`outputXMLURL`*	Specifies the URL of the transformed XML document.
`-XS` *`XSLTURL`*	Specifies the URL of the XSLT stylesheet.

Options

You can supply any number of optional parameters using the `-PARAM` flag. Parameters are specified as name value pairs. The parameter's name must correspond to variables in the XSLT stylesheet. The parameter's value is substituted for the variable when the stylesheet is processed.

Examples

To use the transformer to add an ActiveMQ configured JMS endpoint to an Artix contract you would use a command similar to Example 1.

Example 1: *Using the Transformer*

```
xsltp -XSL oldjmswsdl_to_newjmswsdl.xsl -IN my_old.wsdl -OUT my_new.wsdl -PARAM updateToActiveMQ
   true -PARAM userDefDestinationName dynamicQueues/test.artix.myotherjmstransport
```

CHAPTER 7

Generating Code from WSDL

Artix generates stub and skeleton code that provides a developer with a simple model to develop transport-independent applications.

In this chapter

This chapter discusses the following topics:

C++ Code Generation

Overview

Artix includes a command line tool, `wsdltocpp`, for generating Artix C++ skeletons for the services defined in an Artix contract. It can also generate starting point code for your server and client applications.

WSDLTOCPP

Synopsis

```
wsdltocpp [options] { WSDL-URL | SCHEMA-URL } [-e
web_service_name[:port_list]][-b binding_name][-i port_type]* [-d
output-dir][-n URI=C++namespace]* [-nexclude URI[=C++namespace]]*
[-ninclude URI[=C++namespace]]* [-nimport C++namespace][-impl][-m
{NMAKE | UNIX}:[executable|library]][-libv version][-jp
plugin_class][-f][-server][-client][-sample][-plugin[:plugin_name
]][-deployable][-global][-license][-declspec
declspec][-all][-flags][-upper|-lower|-minimal|-mapper
class][-reflect][-L file][-quiet][-verbose][-h][-v]
```

Parameters

You must specify the location of a valid WSDL contract file, *WSDL-URL*, for the code generator to work.

Options

You can supply the following optional parameters:

`-i port_type`	Specifies the name of the port type for which the tool will generate code. The default is to use the first port type listed in the contract. This switch can appear multiple times.
`-e web_service_nam [:port_list]`	Specifies the name of the service for which the tool will generate code. The default is to use the first service listed in the contract. You can optionally specify a comma separated list of port names to activate. The default is to activate all of the service's ports.
`-b binding_name`	Specifies the name of the binding to use when generating code. The default is the first binding listed in the contract.
`-d output_dir`	Specifies the directory to which the generated code is written. The default is the current working directory.

`-n [URI=]C++namespace`	Maps an XML namespace to a C++ namespace. The C++ stub code generated from the XML namespace, *URI*, is put into the specified C++ namespace, *C++namespace*. This switch can appear multiple times.
`-nexclude URI[=C++namespace]`	Do not generate C++ stub code for the specified XML namespace, *URI*. You can optionally map the XML namespace, *URI*, to a C++ namespace, *C++namespace*, in case it is referenced by the rest of the XML schema/WSDL contract. This switch can appear multiple times.
`-ninclude URI[=C++namespace]`	Generates C++ stub code for the specified XML namespace, *URI*. You can optionally map the XML namespace, *URI*, to a C++ namespace, *C++namespace*. This switch can appear multiple times.
`-nimport C++namespace`	Specifies the C++ namespace to use for the code generated from imported schema.
`-impl`	Generates the skeleton code for implementing the server defined by the contract.
`-m {NMAKE | UNIX} :[executable | library]`	Used in combination with `-impl` to generate a makefile for the specified platform (`NMAKE` for Windows or `UNIX` for UNIX). You can specify that the generated makefile builds an executable, by appending `:executable`, or a library, by appending `:library`. For example, the options, `-impl -m NMAKE:executable`, would generate a Windows makefile to build an executable.
`-libv version`	Used in combination with either `-m NAME:library` or `-m UNIX:library` to specify the version number of the library built by the makefile. This version number is for your own convenience, to help you keep track of your own library versions.
`-f`	*Deprecated*—No longer used (was needed to support routing in earlier versions.
`-server`	Generates code for a sample implementation of a server.
`-client`	Generates code for a sample implementation of a client.

`-sample`	Generates code for a sample implementation of a client and a server (equivalent to `-server -client`).
`-plugin [:plugin_name]`	Generates servant registration code as a Bus plug-in. You can optionally specify the plug-in name by appending `:plugin_name` to this option. If no plug-in name is specified, the default name is `<ServiceName><PortTypeName>`. The service name, `<ServiceName>`, is specified by the `-e` option.
`-deployable`	(Used with `-plugin`.) Generates a deployment descriptor file, `deploy<ServiceName>.xml`, which is needed to deploy a plug-in into the Artix container.
`-global`	(Used with `-plugin`.) In the generated plug-in code, instantiate the plug-in using a `GlobalBusORBPlugIn` object instead of a `BusORBPlugIn` object. A `GlobalBusORBPlugIn` initializes the plug-in automatically, as soon as it is constructed (suitable approach for plug-ins that are linked directly with application code). A `BusORBPlugIn` is not initialized unless the plug-in is either listed in the `orb_plugins` list or deployed into an Artix container (suitable approach for dynamically loading plug-ins).
`-license`	Displays the currently available licenses.
`-declspec declspec`	Creates Visual C++ declaration specifiers for `dllexport` and `dllimport`. This option makes it easier to package Artix stubs in a DLL library.
`-all`	Generate stub code for all of the port types and the types that they use. This option is useful when multiple port types are defined in a WSDL contract.
`-flags`	Displays detailed information about the options.
`-reflect`	Enables reflection on generated data classes.
`-wrapped`	When used with document/literal wrapped style, generates function signatures with wrapped parameters, instead of unwrapping into separate parameters.
`-L file`	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.

`-quiet`	Specifies that the tool runs in quiet mode.
`-verbose`	Specifies that the tool runs in verbose mode.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.

Generated files

The code generator produces a number of stub files from the Artix contract. They are named according to the port type name, *PortTypeName*, specified in the logical portion of the Artix contract. If the contract specifies more than one port type, code will be generated for each one.

The following stub files are generated:

*PortTypeName***.h** defines the superclass from which the client and server are implemented. It represents the API used by the service defined in the contract.

*PortTypeName***Service.h and** *PortTypeName***Service.cxx** are the server-side skeleton code to implement the service defined in the contract.

*PortTypeName***Client.h and** *PortTypeName***Client.cxx** are the client-side stubs for implementing a client to use the service defined by the contract.

*PortTypeName***_wsdlTypes.h and** *PortTypeName***_wsdlTypes.cxx** define the complex datatypes defined in the contract (if any).

*PortTypeName***_wsdlTypesFactory.h and** *PortTypeName***_wsdlTypesFactory.cxx** define factory classes for the complex datatypes defined in the contract (if any).

Java Code Generation

Overview

`wsdltojava` generates JAX-RPC compliant Java code stubs and skeletons for the services defined in the specified Artix contract. It can also generate starting point code for your server and client applicaitons. The default behavior of `wsdltojava` is to generate all of the java code needed to develop a client and server.

WSDLTOJAVA

Synopsis

```
wsdltojava [-e service:port][-b binding][-i portType][-d
output_dir][-p
[namespace=]package][-impl][-server][-client][-plugin][-servlet][
-types][-call][-interface][-sample][-all][-ant][-datahandlers][-m
erge][-deployable][-nexclude namespace[=package]][-ninclude
namespace[=package]][-ser][-L file][-quiet][verbose][-h][-v]
artix-contract
```

Description

You must specify the location of a valid Artix contract for the code generator to work. The default behavior of `wsdltojava` is to generate all of the java code needed to develop a client and server.

Options

You can supply the following optional parameters to control the portions of the code generated:

`-e service:port`	Specifies the name of the service, and optionally the port, for which the tool will generate code. The default is to use the first service listed in the contract. Specifying multiple services results in the generation of code for all the named service/port combinations. If no port is given, all ports defined in a service will be activated.
`-b binding`	Specifies the name of the binding to use when generating code. The default is to use the first binding listed in the contract.

`-i` *`portType`*	Specifies the name of a portType for which code will be generated. You can specify this flag for each portType for which you want code generated. The default is to use the first portType in the contract.
`-d` *`output_dir`*	Specifies the directory to which the generated code is written. The default is the current working directory.
`-p` *`[namespace=]package`*	Specifies the name of the Java package to use for the generated code. You can optionally map a WSDL namespace to a particular package name if your contract has more than one namespace.
`-impl`	Generates the skeleton class for implementing the server defined by the contract.
`-server`	Generates a simple main class for the server.
`-client`	Generates only the Java interface and code needed to implement the complex types defined by the contract. This flag is equivalent to specifying `-interface -types`.
`-plugin`	Generate a bus plug-in with the appropriate servant registration code for the generated service implementation.
`-servlet`	Generates a bus plug-in with the additional information needed to deploy it as a servlet.
`-types`	Generates the code to implement the complex types defined by the contract.
`-call`	Generates a sample client the uses the `Call` interface to invoke on the remote service.
`-interface`	Generates the Java interface for the service.
`-sample`	Generates a sample client that can be used to test your Java server.
`-all`	Generates code for all portTypes in the contract.
`-ant`	Generate an ant build target for the generated code.

`-datahandlers`	When a service uses SOAP w/ attachments as its payload format, generate code that uses `javax..activation.DataHandler` instead of the standard Java classes specified in the JAX-RPC specification.
`-merge`	Merge any user changes into the generated code.
`-deployable`	Generate a deployment descriptor to deploy the generated plug-in into an Artix container. For more information see Deploying and Managing Artix Solutions.
`-nexclude` *`namespace[=package]`*	Instructs the code generator to skip the specified XMLSchema namespace when generating code. You can optionally specify a package name to use for the types that are not generated.
`-ninclude` *`namespace[=package]`*	Instructs the code generator to generate code for the specified XMLSchema namespace. You can optionally specify a package name to use for the types in the specified namespace.
`-ser`	Specifies that the generated classes for the types defined in a contract should be serializable.
`-stub`	Specifies that the tool will generate the stub code for a client and a server.
`-L` *`file`*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-quiet`	Specifies that the tool runs in quiet mode.
`-verbose`	Specifies that the tool runs in verbose mode.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.

Generated files

The Artix code generator produces a number of files from the Artix contract. They are named according to the port name specified when the code was generated. The files include:

portTypeName.**java** defines the Java interface that both the client and server implement.

*portTypeName***Impl.java** defines the class used to implement the server.

*portTypeName***Server.java** is a simple main class for the server.

In addition to these files, the code generator also creates a class for each named schema type defined in the Artix contract. These files are named according to the type name they are given in the contract and contain the helper functions needed to use the data types. The naming convention for the helper type functions conforms to the JAX-RPC specification.

Generated type packages

The generated types are generated into a single package which must be imported for any methods using them. By default, the package name will be mapped from the target namespace of the schema describing the types. The default package name is created following the algorithm specified in the JAXB specification. The mapping algorithm follows four basic steps:

1. The leading `http://` or `urn://` are stripped off the namespace.
2. If the first string in the namespace is a valid internet domain, for example it ends in `.com` or `.gov`, the leading `www.` is stripped off the string, and the two remaining components are flipped.
3. If the final string in the namespace ends with a file extension of the pattern `.xxx` or `.xx`, the extension is stripped.
4. The remaining strings in the namespace are appended to the resulting string and separated by dots.
5. All letters are made lowercase.

For example, the XML namespace `http://www.widgetVendor.com/types/widgetTypes.xsd` would be mapped to the Java package name `com.widgetvendor.types.widgettypes`.

Exceptions

If you generate code from a WSDL file that contains multiple portTypes, multiple bindings, multiple services, or multiple ports `wsdltojava` will generate a warning message informing you that it is using the first instance of each to use for generating code. If you use the command line flags to specify which instances to use, the warning message is not displayed.

Database Intermediary Generation

Overview

The `wsdltodbservice` tool takes a WSDL document and an Artix database configuration document and generates the code for the intermediary used expose the database operations. The generated Java code will need to be compiled before it can be deployed.

WSDLTODBSERVICE

Synopsis

```
wsdltodbservice [-d dir][-source
dir][-plugin][-h][-v][-quiet][verbose] dbconfig wsdl
```

Options

The tool has the following options:

`-d dir`	Specifies the output directory for the generated DB service.
`-source dir`	Specifies the output directory for the generated source code. The default is `java`.
`-plugin`	Specifies that the DB service is to be generated as a plug-in for deployment into an Artix container.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.
`-quiet`	Specifies that the tool runs in quiet mode.
`-verbose`	Specifies that the tool runs in verbose mode.

CHAPTER 8

Tools for Generating Support Files

Artix provides a tools to generate a number of support files that can be used in conjunction with Artix solutions.

In this chapter

This chapter discusses the following topics:

Generating IDL from WSDL

Overview

The `wsdltocorba` tool compiles Artix contracts containing a CORBA binding and generates IDL for the specified binding and port type.

WSDLTOCORBA

Synopsis

```
wsdltocorba -idl -b binding [-corba] [-i portType] [-d dir] [-o
file][-L file][-quiet][-verbose][-h][-v] wsdl_file
```

Parameters

The command has the following required parameters:

`-idl`	Instructs the tool to generate an IDL file from the specified binding.
`-b binding`	Specifies the CORBA binding from which to generate IDL.
`wsdl_file`	Specifies the WSDL file to process.

Options

The command has the following options:

`-corba`	Instructs the tool to generate a CORBA binding for the specified port type.
`-i portType`	Specifies the name of the port type being mapped to a CORBA binding.
`-d dir`	Specifies the directory into which the new WSDL file is written.
`-o file`	Specifies the name of the generated WSDL file. Defaults to *`wsdl_file`*`.idl`.
`-L file`	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-quiet`	Specifies that the tool runs in quiet mode.
`-verbose`	Specifies that the tool runs in verbose mode.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.

Notes

By combining the `-idl` and `-corba` flags with `wsdltocorba`, you can generate a CORBA binding for a logical operation and then generate the IDL for the

generated CORBA binding. When doing so, you must also use the `-i` *`portType`* flag to specify the port type from which to generate the binding and the `-b` *`binding`* flag to specify the name of the binding to from which to generate the IDL.

Generating a Deployment Descriptor

Overview

The `wsdd` tool generates a deployment descriptor that can be used to deploy and Artix plug-in into the Artix container.

WSDD

Synopsis

```
wsdd -service QName -pluginName name -pluginType {Cxx|Java}
[-pluginImpl name][-pluginURL dir][-wsdlurl URL][-provider
namespace][-file file][-d dir][-verbose][-q][-h][-v]
```

Parameters

The command has the following required parameters:

`-service` *`QName`*	Specifies the QName of the plug-in's service as given in its contract.
`-pluginName` *`name`*	Specifies the name of the plug-in as specified in the Artix configuration file.
`-pluginType {Cxx|Java}`	Specifies if the plug-in is implemented in C++ or Java.

Options

The command has the following options:

`-pluginImpl` *`name`*	Specifies the library/class name of the plug-in's implementation.
`-pluginURL` *`dir`*	Specifies the directory where the plug-in's implementation is located.
`-wsdlurl` *`URL`*	Specifies the location of the contract defining the service implemented by the plug-in.
`-provider` *`namespace`*	Specifies the namespace under which your plug-in's `ServantProvider` is registered with the bus.
`-file` *`file`*	Specifies the name of the generated deployment descriptor.
`-d` *`dir`*	Specifies the directory where the generated file will be written.
`-verbose`	Specifies that the tool runs in verbose mode.
`-quiet`	Specifies that the tool runs in quiet mode.

`-h`	Displays the tool's usage message.
`-v`	Displays the tool's version.

Generating an ACL File

Overview

The `wsdltoacl` tool generates an ACL file for the operation for which the default role name is not sufficient. It takes a WSDL file and generates an appropriate ACL file. You will need to add information specific to your deployment to this file.

WSDLTOACL

Synopsis

```
wsdltoacl -s server WSDL-URL [-i interface][-r default_role][-d
output_dir][-o output_file][-props props_file][-L
file][-quiet][-verbose][-h][-v]
```

Parameters

The command has the following required parameters:

`-s server`	Specifies the name of the server. Typically this is the ORB name of the server.
`WSDL-URL`	Specifies the name of the WSDL file from which the ACL file is generated.

Options

The command has the following options:

`-i interface`	Specifies the `<portType>` for which ACL data will be generated. The default is to generate information for all port types defined in the contract.
`-r default_role`	Specifies the role name to use in the generated ACL document. The default is `IONAUserRole`.
`-d output_dir`	Specifies the directory where the generated file will be written.
`-o output_file`	Specifies the name of the generated ACL file. The default is to use the name of the WSDL file with a `.acl` extension.
`-props props_file`	Specifies the properties file listing the roles for each operation.
`-L file`	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.

`-quiet`	Specifies that the tool runs in quiet mode.
`-verbose`	Specifies that the tool runs in verbose mode.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.

Index

W

X