PROGRESS SOFTWARE

Artix® ESB

Artix® ESB Command Reference

Version 5.5
December 2008

Artix® ESB Command Reference

Version 5.5

Published 01 Dec 2008

Legal Notices

Table of Contents

List of Examples

Preface

What is Covered in This Book

This book is a reference to the command line tools included with Artix ESB. The command line tools replicates the functionality of the GUI.

Who Should Read This Book

This book is intended for developers who use command line tools as part of their build and development environments. However, all users of Artix ESB can benefit from using this as a reference.

The Artix ESB Documentation Library

For information on the organization of the Artix ESB library, the document conventions used, and where to find additional resources, see Using the Artix ESB Library [http://www.iona.com/support/docs/artix/5.1/library_intro/index.htm].

Using the Artix Tools

*For the most commonly used tools, Artix ESB provides a universal access point through the **artix** tool. However, legacy tools, such as the JAX-RPC code generators, are only accessible by explicitly calling them.*

Overview

Artix ESB Java Runtime and Artix ESB C++ Runtime share a number of command line tools. To make it easier to determine which version of the tools you are using, Artix has integrated a number of the tools into a single tool. The new tool, **artix** provides access to the the C++ and JAX-WS code generators, the Artix ESB Java Runtime tool for adding a service to a WSDL document, the Artix ESB Java Runtime CORBA tools, and the DB service tools.

The Artix ESB C++ Runtime version of the tools and the JAX-RPC code generators can be called by explicitly calling them.

The artix tool

The **artix** tool provides access to the following tools:

- **wsdl2java**

Warning

This is the JAX-WS version of the Java code generator and generates code the only runs using Artix ESB Java Runtime.

- **wsdl2cpp**
- **java2ws**

Warning

This tool only works with JAX-WS classes that can be run using Artix ESB Java Runtime.

- **wsdlgen**
- **validator**
- **idl2wsdl**

Warning

This version of the tool generates WSDL that is only usable by Artix ESB Java Runtime.

- **wsdl2idl**

Warning

This version of the tool only works with WSDL containing a Artix ESB Java Runtime CORBA binding.

- **wsdl2service**

Important

This version of the tool only generates SOAP and JMS endpoints.

- **wsdl2soap**
- **wsdl2xml**
- **xsd2wsdl**
- **wsdl2dbservice**
- **dbconfig2wsdl**
- **wsdlgen**
- **cobol2brgeinfo**
- **idl2cobol**
- **idl2pli**
- **wsdl2pli**
- **wsdl2cobol**
- **transformerstore**
- **wsdl2acl**

- **sql2dbconfig**

You can retrieve this list by using the following command:

```
artix
```

You can view the options for each of the tools using the following command:

```
artix tool
```

Other tools

The JAX-RPC version of the Java code generator, the JAX-RPC version of the WSDL generator, and the Artix ESB C++ Runtime versions of the IDL tools are still available as part of Artix ESB. You can use them, as well as the mainframe tools, by explicitly calling them from the command line and specifying the full path to the tool.

The Artix ESB C++ Runtime version of the tools are located in *InstallDir*/cxx_java/bin.

The mainframe tools are located in *InstallDir*/mainframe/bin.

Prerequisites

Before you can use these tools you must properly set up your environment. How you set up your environment depends on which runtime you are intending to use.

To set up your environment to use Artix ESB C++ Runtime do the following:

1. Run the **artix_env** script located in *InstallDir*/cxx_java/bin.
2. If you are going to be developing JAX-RPC applications, ensure that JAVA_HOME is pointing to a valid Java 5(or higher) JDK.

Warning

The Artix installer only installs a JRE.

To set up your environment to use Artix ESB Java Runtime do the following:

1. Run the **artix_java_env** script located in *InstallDir*/java/bin.
2. Ensure that JAVA_HOME points to a Java 5 (or higher) JDK.

Warning

The Artix installer only installs a JRE.

Generating WSDL

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

Name

artix java2ws — generates WSDL and other artifacts from JAX-WS compliant Java code

Synopsis

`artix java2ws` [[-?] | [-help] | [-h]] [-frontend { jaxws | simple }] [-databinding { jaxb | aegis }] [-wsdl] [-wrapperbean] [-client] [-server] [-ant] [-o *`outFile`*] [-s *`sourceDir`*] [-d *`resourceDir`*] [-classdir *`classDir`*] [-cp *`classpath`*] [-soap12] [-t *`targetNamespace`*] [-beans *`beanPath`*...] [-servicename *`serviceName`*] [-portname *`portName`*] [-createxsdimports] [-v] [[-verbose] | [-quiet]] *`classname`*

Description

artix java2ws takes a service endpoint implementation (SEI) and generates the support files used to implement a Web service. **artix java2ws** can generate the following:

- a WSDL document
- the server code needed to deploy the service as a POJO
- client code for accessing the service
- wrapper and fault beans

Arguments

The arguments used to manage the code generation process are reviewed in the following table.

Option	Interpretation
`-?`	Displays the online help for this utility.
`-help`	
`-h`	

Option	Interpretation
`-frontend {jaxws|simple}`	Specifies front end to use for processing the SEI and generating the support classes. `jaxws` is the default.
`-databinding {jaxb|aegis}`	Specifies the data binding used for processing the SEI and generating the support classes. The default when using the JAX-WS front end is `jaxb`. The default when using the simple frontend is `aegis`.
`-wsdl`	Instructs the tool to generate a WSDL document.
`-wrapperbean`	Instructs the tool to generate the wrapper bean and the fault beans.
`-client`	Instructs the tool to generate client code.
`-server`	Instructs the tool to generate server code.
`-ant`	Instructs the tool to generate an Ant build script to compile the generated code.
`-o` *outFile*	Specifies the name of the generated WSDL file.
`-s` *sourceDir*	Specifies the directory into which the generated source files are placed.
`-d` *resourceDir*	Specifies the directory into which the resource files are placed.
`-classdir` *classDir*	Specifies the directory into which the generated source files are compiled. If this option is not used, the generated source is not compiled.
`-cp` *classpath*	Specifies the classpath searched when processing the SEI.
`-soap12`	Specifies that the generated WSDL document is to include a SOAP 1.2 binding.
`-t` *targetNamespace*	Specifies the target namespace to use in the generated WSDL file.
`-beans` *beanPath*	Specifies the path used to locate the bean definition files.
`-servicename` *serviceName*	Specifies the value of the generated `service` element's `name` attribute.
`-portname` *portName*	Specify the value of the generated `port` element's `name` attribute.
`-createxsdimports`	Instructs the tool to generate a separate schema for the types instead of including the types directly in the generated WSDL document.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.

Option	Interpretation
classname	Specifies the name of the SEI class.

Using Ant

To call this tool from Ant you execute the `org.apache.cxf.tools.java2ws.JavaToWS` class.

Example 1 on page 18 shows the **java** task to generate WSDL from an SEI.

Example 1. Generating WSDL From Ant

```
<java classname="org.apache.cxf.tools.java2ws.JavaToWS"
fork="true">
  <arg value="-wsdl"/>
  <arg value="Service.greeter"/>
  <classpath>
    <path refid="fsf.classpath"/>
  </classpath>
</java>
```

Name

javatowsdl — generates a WSDL document from a JAX-RPC compliant Java class

Synopsis

`javatowsdl` [-h] [-o *`output-file`*] [-t *`target-namespace`*] [-x *`schema-namespace`*] [-i *`portType`*] [-useTypes] [-qualified] [-L *`license`*] [-v] [[-verbose] | [-quiet]] *`classname`*

Description

javatowsdl uses the mapping rule described in Sun's JAX-RPC 1.1 specification to generate a WSDL file from a Java class.

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.

Warning

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.

Arguments

The tool has the following required arguments:

Option	Interpretation
`classname`	Specifies the name of the Java class.

Arguments

The tool has the following optional arguments:

Option	Interpretation
`-h`	Displays the online help for this utility.
`-o` *`output-file`*	Specifies the name of the generated WSDL file.
`-t` *`target-namspace`*	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` *`schema-namespace`*	Specifies the target namespace of the XML Schema information generated to represent the data types inside the WSDL document.By default, the generated target namespace of the XML Schema 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.
`-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..
`-L` *`license`*	Specifies the location of your Artix ESB license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.

Name

artix idl2wsdl — generates an Artix ESB Java Runtime compliant WSDL document from a CORBA IDL file

Synopsis

`artix idl2wsdl` [-I *`idl-include-dir`*...] [-o *`output-dir`*] [-a *`corba-address`*] [-b] [-f *`corba-address-file`*] [-n *`schema-import-file`*] [-s *`idl-sequence-type`*] [-w *`target-namespace`*] [-x *`schema-namespace`*] [-t *`corba-typemap-namespace`*] [-L *`logical-wsdl-filename`*] [-P *`physical-wsdl-filename`*] [-T *`schema-filename`*] [-qualified] [-e *`xml-encoding-type`*] [-mns*`namespaceMapping`*] [-ow *`wsdloutput-file`*] [ex*`excludedModules`*] [-pf] [-v] [[-verbose] | [-quiet]] *`idl`*

Description

artix idl2wsdl supports several options that control the generation of 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.

Required Arguments

The command has the following required arguments:

Option	Interpretation
`idl`	Specifies the name of the IDL file.

Optional Arguments

The command has the following optional arguments:

Option	Interpretation
`-I` *`idl-include-dir`*	Specify a directory to be included in the search path for the IDL preprocessor. You can use this flag multiple times.

Option	Interpretation
`-o output-dir`	Specifies the directory into which the WSDL file is written.
`-a corba-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.
`-b`	Specifies that bounded strings are to be treated as unbounded. This eliminates the generation of the special types for the bounded string.
`-f corba-address-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.
`-n schema-import-file`	Specifies that a schema file is to be included in the generated contract by an import statement. This option cannot be used with the `-T` option.
`-s idl-sequence-type`	Specifies the XML Schema type used to map the IDL sequence<octet> type. Valid values are `base64Binary` and `hexBinary`. The default is `base64Binary`.
`-w target-namespace`	Specifies the namespace to use for the WSDL document's target namespace.
`-x schema-namespace`	Specifies the namespace to use for the generated XML Schema's target namespace.
`-t corba-typemap-namespace`	Specifies the namespace to use for the CORBA type map's target namespace.
`-L logical-wsdl-filename`	Specifies that the logical portion of the generated WSDL specification into is written to *`logical-wsdl-filename`*. The logical WSDL is then imported into the default generated file.
`-P physical-wsdl-filename`	Specifies that the physical portion of the generated WSDL specification into is written to *`physical-wsdl-filename`*. The physical WSDL is then imported into the default generated file.
`-T schema-filename`	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.
`-qualified`	Generates fully qualified WSDL.
`-e xml-encoding-type`	Specifies the value for the generated WSDL document's xml encoding attribute. The default is UTF-8.
`-mns`*`namespaceMapping`*	Specifies a mapping between IDL modules and XML namespaces.
`-ow wsdloutput-file`	Specifies the name of the generated WSDL file.

Option	Interpretation
`-ex`*`excludeModules`*	Specifies one or more IDL modules to exclude when generating the WSDL file.
`-pf`	Specifies that polymorphic factory support is enabled.
`-h`	Displays the tool's usage statement.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.

Name

idltowsdl — generates an Artix ESB C++ Runtime compliant WSDL document from a CORBA IDL file

Synopsis

`idltowsdl` [-usetypes] [-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`*] [-h] [-v] [[-quiet] | [-verbose]] *`idlfile`*

Description

idltowsdl 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.

Required Arguments

The command has the following required arguments:

Option	Interpretation
`idlfile`	Specifies the name of the IDL file.

Optional Arguments

The command has the following optional arguments:

Option	Interpretation
`-usetypes`	Generate rpc style messages. rpc style messages have parts defined using XML Schema 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.

Option	Interpretation
`-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 prior to running the command.
`-o dir`	Specifies the directory into which the WSDL file is written.
`-s type`	Specifies the XML Schema 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 is written to *`file`*. *`file`* is then imported into the default generated file.
`-P file`	Specifies that the physical portion of the generated WSDL is written to *`file`*. *`file`* is then imported into the default generated file.
`-w namespace`	Specifies the namespace to use for the WSDL document's target namespace. The default is `http://schemas.iona.com/idl/`*`idl_name`*.
`-x namespace`	Specifies the namespace to use for the generated XML Schema's target namespace. The default is `http://schemas.iona.com/idltypes/`*`idl_name`*.
`-t namespace`	Specifies the namespace to use for the CORBA type map's target namespace. 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 cannot be used with `-n`.
`-n file`	Specifies that a schema file, *`file`*, is to be included in the generated contract by an import statement. This cannot be used with `-T`.
`-b`	Specifies that bounded strings are to be treated as unbounded. This eliminates the generation of the special types for the bounded strings.
`-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.

Option	Interpretation
`-3`	Use relaxed IDL grammar checking semantics to allow IDL used by Orbix 3 to be parsed.
`-fastrack`	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`.
`-h`	Displays the tool's usage statement.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.

Name

artix cobol2wsdl — generates a WSDL document with a fixed binding from a COBOL copybook

Synopsis

`artix cobol2wsdl` {-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]

Required Arguments

The command has the following required arguments:

Option	Interpretation
`-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.

Optional Arguments

The command has the following optional arguments:

Option	Interpretation
`-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.

Option	Interpretation
`-fm [faultmessage:]faultcopybook`	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 license file. The default is `IT_PRODUCT_DIR\etc\license.txt`.
`-h`	Displays the tool's usage statement.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.

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

Name

artix xsd2wsdl — generates a WSDL document containing the types defined in an XML Schema document.

Synopsis

`artix xsd2wsdl` [[-?] | [-help] | [-h]] [-t *`target-namespace`*] [-n *`wsdl-name`*] [-d *`output-directory`*] [-o *`output-file`*] [-v] [[-verbose] | [-quiet]] {*`xsdurl`*}

Description

artix xsd2wsdl imports an XML Schema document and generates a WSDL file containing a `types` element populated by the types defined in the XML Schema document.

Required Arguments

The command has the following required arguments:

Option	Interpretation
`-t` *`target-namespace`*	Specifies the target namespace for the generated WSDL.
`xsdurl`	The path and name of the existing XSDSchema file.

Optional Arguments

The command has the following optional arguments:

Option	Interpretation
`-?`	Displays the online help for this utility.
`-help`	
`-h`	
`-n` *`wsdl-name`*	Specifies the value of the generated `definition` element's `name` attribute.
`-d` *`output-directory`*	Specifies the directory in which the generated WSDL is placed.

Option	Interpretation
`-o output-file`	Specifies the name of the generated WSDL file.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.

Using Ant

To call this tool from Ant you execute the `org.apache.cxf.tools.misc.XSDToWSDL` class.

Example 2 on page 30 shows the **java** task to execute this command.

Example 2. Generating a WSDL from a Schema Using Ant

```
<java classname="org.apache.cxf.tools.misc.XSDToWSDL"
fork="true">
  <arg value="-t"/>
  <arg value="http://cxf.apache.org/demos"/>
  ...
  <arg value="MyXSD.xsd"/>
  <classpath>
    <path refid="fsf.classpath"/>
  </classpath>
</java>
```

Name

xsdtowsdl — generates a WSDL document containing the types defined in an XML Schema document

Synopsis

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

Description

xsdtowsdl imports an XML Schema document and generates a WSDL contract containing a types element populated by the types defined in the XML Schema document. The rest of the contract will be empty.

Arguments

The arguments used to manage the WSDL file generation are reviewed in the following table.

Option	Interpretation
`-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 license file. The default is `IT_PRODUCT_DIR\etc\license.txt`.
`-h`	Displays the tool's usage statement.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.

Option	Interpretation
`-quiet`	Suppresses comments during the code generation process.
`xsdurl`	Specifies the URL of the XML Schema document.

Name

artix dbconfig2wsdl — generates a WSDL document from an Artix ESB Java Runtime database configuration file

Synopsis

`artix dbconfig2wsdl` [-a *`address`*] [-d *`output-dir`*] [-servicename *`service-name`*] [-jdbctypemappings *`jdbc-type-mapping-file`*] [-mp { element | type }] [-t *`target-namespace`*] [-o *`output-file`*] [-logical] [-v] [[-verbose] | [-quiet]] *`dbconfigurl`*

Description

artix dbconfig2wsdl imports an Artix ESB Java Runtime database configuration document and generates an Artix ESB Java Runtime contract defining a service that represents the database operations defined in the document.

Required Arguments

The tool has the following required arguments:

Option	Interpretation
`dbconfigurl`	Specifies the URL of the database configuration file.

Optional Arguments

The tool has the following optional arguments:

Option	Interpretation
`-a` *`address`*	Specifies the value of generated `soap:address` element's `location` attribute.
`-d` *`output-dir`*	Specifies the folder into which the generated WSDL is placed.
`-servicename` *`name`*	Specifies the value of the generated `service` element's `name` attribute. The default is `DataService`.
`-jdbctypemappings` *`jdbc-type-mapping-file`*	Specifies the name of the file containing the mappings between JDBC types and XSD types.

Option	Interpretation
`-mp {element | type}`	Specifies if the generated message parts should be types or elements. The default is elements.
`-t` *`target-namespace`*	Specifies the target namespace for the generated contract.
`-o` *`output-file`*	Specifies the name of the generated WSDL document.
`-logical`	Specifies the tool only generates the logical portion of the WSDL document.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.

Name

dbconfigtowsdl — generates a WSDL document from an Artix ESB C++ Runtime database configuration file

Synopsis

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

Description

dbconfigtowsdl imports an Artix ESB C++ Runtime database configuration document and generates an Artix ESB C++ Runtime contract defining a service that represents the database operations defined in the document.

Required Arguments

The command has the following required arguments:

Option	Interpretation
`dbconfigurl`	Specifies the URL of the database configuration file.

Optional Arguments

The command has the following optional arguments:

Option	Interpretation
`-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 ESB C++ Runtime plug-in. This flag is only valid when `-fastttrack` is also used.

Option	Interpretation
`-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 version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.

Adding Bindings

Artix provides command line tools for adding SOAP, XML, and CORBA bindings to WSDL documents.

Name

artix wsdl2soap — generates a WSDL document containing a valid SOAP/HTTP endpoint definition based on a `portType` element.

Synopsis

`artix wsdl2soap` [[-?] | [-help] | [-h]] {-i *`port-type-name`*} [-b *`binding-name`*] [-soap12] [-d *`output-directory`*] [-o *`output-file`*] [-n *`soap-body-namespace`*] [-style { document | rpc }] [-use (literal/encoded)] [-v] [[-verbose] | [-quiet]] *`wsdlurl`*

Description

artix wsdl2soap will generate a new WSDL file with a SOAP binding from an existing WSDL file containing a `portType` element.

Required Arguments

The command has the following required arguments:

Option	Interpretation
`-i` *`port-type-name`*	Specifies the `portType` element for which a binding should be generated.
`wsdlurl`	The path and name of the WSDL file containing the `portType` element definition.

Optional Arguments

The command has the following optional arguments:

Option	Interpretation
`-?`	Displays the online help for this utility.
`-help`	
`-h`	
`-b` *`binding-name`*	Specifies the name of the generated SOAP binding.

Option	Interpretation
`-soap12`	Specifies that the generated binding will use SOAP 1.2.
`-d` *`output-directory`*	Specifies the directory to place generated WSDL file.
`-o` *`output-file`*	Specifies the name of the generated WSDL file.
`-n` *`soap-body-namespace`*	Specifies the SOAP body namespace when the style is RPC.
`-style` (`document`/`rpc`)	Specifies the encoding style (document or RPC) to use in the SOAP binding. The default is `document`.
`-use` (`literal`/`encoded`)	Specifies the binding use (encoded or literal) to use in the SOAP binding. The default is `literal`.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.

If the `-style rpc` argument is specified, the `-n` *`soap-body-namspace`* argument is also required. All other arguments are optional and may be listed in any order.

Using Ant

To call this tool from Ant you execute the `org.apache.cxf.tools.misc.WSDLToSoap` class.

Example 3 on page 39 shows the **java** task to generate a SOAP 1.2 binding.

Example 3. Generating a SOAP 1.2 Binding From Ant

```
<java classname="org.apache.cxf.tools.misc.WSDLToSoap"
fork="true">
  <arg value="-i"/>
  <arg value="greeter"/>
  <arg value="-soap12"/>
  ...
  <arg value="MyWSDL.wsdl"/>
  <classpath>
    <path refid="fsf.classpath"/>
  </classpath>
</java>
```

Name

artix wsdl2xml — generates a WSDL document containing an XML binding based on a `portType` element.

Synopsis

`artix wsdl2xml` [[-?] | [-help] | [-h]] [-i *`port-type-name`*] [-b *`binding-name`*] [-e *`service-name`*] [-p *`port-name`*] [-a *`address`*] [-d *`output-directory`*] [-o *`output-file`*] [-v] [[-verbose] | [-quiet]] {*`wsdlurl`*}

Description

artix wsdl2xml generates an XML binding from an existing WSDL document containing a `portType` element.

Arguments

The arguments used to manage WSDL file generation are reviewed in the following table.

Option	Interpretation
`-i` *`port-type-name`*	Specifies the `portType` element to use.
`wsdlurl`	The path and name of the existing WSDL file.

Optional Arguments

The command takes the following optional arguments:

Option	Interpretation
`-?`	Displays the online help for this utility.
`-help`	
`-h`	
`-b` *`binding-name`*	Specifies the name of the generated XML binding.
`-e` *`service-name`*	Specifies the value of the generated `service` element's `name` attribute.

Option	Interpretation
`-p port-name`	Specifies the value of the generated `port` element's `name` attribute. To specify multiple `port` elements, separate the names by a space.
`-a address`	Specifies the value used in the `address` element of the generated `port` element.
`-d output-directory`	Specifies the directory to place generated WSDL file.
`-o output-file`	Specifies the name of the generated WSDL file.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.

Using Ant

To execute this tool using Ant set the **java** task's classname property to `org.apache.cxf.tools.misc.WSDLToXML`.

Example 4 on page 41 shows the **java** task to execute this command.

Example 4. Generating a SOAP Binding From Ant

```
<java classname="org.apache.cxf.tools.misc.WSDLToXML"
fork="true">
  <arg value="-i"/>
  <arg value="greeter"/>
  ...
  <arg value="MyWSDL.wsdl"/>
  <classpath>
    <path refid="artix_java.classpath"/>
  </classpath>
</java>
```

Name

artix wsdl2idl -corba — adds an Artix ESB Java Runtime CORBA binding to a WSDL document

Synopsis

`artix wsld2idl` {-corba} {-i *`portType`*} [-idl] [-b *`binding`*] [-d *`dir`*] [-w *`wsdlOut`*] [-o *`idlOut`*...] [-props *`namespace`*] [-wrapped] [-a *`address`*] [-f *`address-file`*] [[-quiet] | [-verbose]] [-v] [-h] *`wsdl`*

Description

artix wsdl2idl -corba adds a Artix ESB Java Runtime CORBA binding to an existing WSDL document. The generated WSDL file will also contain a Artix ESB Java Runtime CORBA port with no address specified.

Tip

You can also generate an IDL file that corresponds to the generated CORBA binding by using the `-idl` option.

Required Arguments

The tool has the following required arguments:

Option	Interpretation
`-corba`	Specifies that the tool will generate a new WSDL file with a CORBA binding.
`-i` *`portType`*	Specifies the name of the interface for which the CORBA binding is generated.
`wsdl`	Specifies the WSDL document to which the binding is added.

Optional Arguments

The tool has the following optional arguments:

Option	Interpretation
`-idl`	Specifies that an IDL file will be generated for the generated CORBA binding. You must also use the `-b` flag in conjunction with this flag.
`-b` *`binding`*	Specifies the name of the generated CORBA binding.
`-d` *`dir`*	Specifies the directory into which the new WSDL document is written.
`-w` *`wsdlOut`*	Specifies the name of the WSDL document containing the generated CORBA binding.
`-o` *`idlOut`*	Specifies the name of the generated IDL file.
`-props` *`namespace`*	Specifies the namespace to use for the generated CORBA typemap.
`-wrapped`	Specifies that the generated binding uses wrapped types.
`-a` *`address`*	Specifies the value of the generated binding's `corba:address` element's `location` attribute.
`-f` *`address-file`*	Specifies the name of a file whose contents are to be used as the value of the generated binding's `corba:address` element's `location` attribute.
`-v`	Displays the tool's version.
`-h`	Specifies that the tool will display a detailed usage statement.
`-quiet`	Specifies that the tool is to run in quiet mode.
`-versbose`	Specifies that the tool is to run in verbose mode.

Name

wsdltosoap — generates a WSDL document containing an Artix ESB C++ Runtime SOAP binding

Synopsis

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

Description

wsdltosoap adds a Artix ESB C++ Runtime SOAP binding to a WSDL document based on the values provided as arguments to the tool.

Required Arguments

The tool has the following required arguments:

Option	Interpretation
`-i` *`portType`*	Specifies the name of the `portType` element being mapped to a SOAP binding.
`-n` *`namespace`*	Specifies the namespace to use for the SOAP binding.
`wsdlurl`	Specifies the WSDL document from which to base the generated WSDL document.

Optional Arguments

The tool has the following optional arguments:

Option	Interpretation
`-soapversion {1.1 | 1.2}`	Specifies the SOAP version of the generated binding. Defaults to `1.1`.
`-style {document | rpc}`	Specifies the encoding style to use in the SOAP binding. Defaults to `document`.
`-use {literal | encoded}`	Specifies how the data is encoded. Default is `literal`.
`-o` *`file`*	Specifies the filename for the generated contract. The default is to append `-service` to the name of the imported contract.

Option	Interpretation
`-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.

Name

wsdltocorba -corba — adds an Artix ESB C++ Runtime CORBA binding to a WSDL document

Synopsis

`wsdltocorba -corba` {-i *portType*} [-idl] [-d *dir*] [-b *binding*] [-o *file*] [-props *namespace*] [-wrapped] [-L *file*] [[-quiet] | [-verbose]] [-h] [-v] *wsdl*

Description

wsdltocorba -corba adds a Artix ESB C++ Runtime CORBA binding to an existing WSDL document. The generated WSDL file will also contain a Artix ESB C++ Runtime CORBA port with no address specified.

Tip

You can also generate an IDL file that corresponds to the generated CORBA binding by using the `-idl` option.

Required Arguments

The tool has the following required arguments:

Option	Interpretation
`-i` *portType*	Specifies the name of the port type for which the CORBA binding is generated.
wsdl	Specifies the WSDL document to which the binding is added.

Optional Arguments

The tool has the following optional arguments:

Option	Interpretation
`-idl`	Specifies that an IDL file will be generated for the generated CORBA binding. You must also use the `-b` flag in conjunction with this flag.
`-d` *dir*	Specifies the directory into which the new WSDL document is written.

Option	Interpretation
-b *binding*	Specifies the name of the generated CORBA binding. The default is *portType*Binding.
-o *file*	Specifies the name of the generated WSDL document. The default is *wsdl_file*-corba.wsdl.
-props *namespace*	Specifies the namespace to use for the generated CORBA typemap.
-wrapped	Specifies that the generated binding uses wrapped types.
-L *file*	Specifies the location of your Artix license file. The default behavior is to check IT_PRODUCT_DIR\etc\license.txt.
-h	Displays the tool's usage statement.
-v	Displays the tool's version.
-quiet	Specifies that the tool is to run in quiet mode.
-versbose	Specifies that the tool is to run in verbose mode.

Adding Endpoints

Artix provides command line tools for adding endpoints to WSDL documents.

Name

artix wsdl2service -transport http — generates a WSDL document containing a valid HTTP endpoint definition from a `binding` element.

Synopsis

`artix wsdl2service - transport http` [[-?] | [-help] | [-h]] [-e *`service-name`*] [-p *`port-name`*] { -n *`binding-name`*} [-a *`address`*] [-soap12] [-o *`output-file`*] [-d *`output-directory`*] [-v] [[-verbose] | [-quiet]] { *`wsdlurl`* }

Description

artix wsdl2service -transport http creates a new WSDL file containing an HTTP service definition from an existing WSDL document containing a binding element.

Arguments

The arguments used to manage the WSDL file generation are reviewed in the following table.

Option	Interpretation
`-?`	Displays the online help for this utility.
`-help`	
`-h`	
`-e` *`service-name`*	Specifies the value of the generated `service` element's `name` attribute.
`-p` *`port-name`*	Specifies the value of the generated `port` element's `name` attribute. To specify multiple port elements, separate the names by a space.
`-a` *`address`*	Specifies the value used in the `address` element of the port.
`-soap12`	Specifies that the SOAP version to use is 1.2.
`-n` *`binding-name`*	Specifies the binding used to generate the service.
`-o` *`output-file`*	Specifies the name of the generated WSDL file.

Option	Interpretation
`-d output-directory`	Specifies the directory in which the generated WSDL is placed.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.
wsdlurl	The path and name of the existing WSDL file.

Using Ant

To call this tool from Ant you execute the `org.apache.cxf.tools.misc.WSDLToService` class.

Example 5 on page 51 shows the **java** task to generate a HTTP binding.

Example 5. Generating a JMS Binding From Ant

```
<java classname="org.apache.cxf.tools.misc.WSDLToService"
fork="true">
  <arg value="-transport"/>
  <arg value="http"/>
  <arg value="-n"/>
  <arg value="JMSSoapBinding"/>
  ...
  <arg value="MyWSDL.wsdl"/>
  <classpath>
    <path refid="fsf.classpath"/>
  </classpath>
</java>
```

Name

artix wsdl2service -transport jms — generates a WSDL document containing a valid JMS endpoint definition from a `binding` element.

Synopsis

`artix wsdl2service -transport jms` [[-?] | [-help] | [-h]] [-e *`service-name`*] [-p *`port-name`*] { -n *`binding-name`*} [[-jds (queue/topic)] | [-jpu *`jndi-provider-URL`*] | [-jcf *`initial-context-factory`*] | [-jfn *`jndi-connection-factory-name`*] | [-jdn *`jndi-destination-name`*] | [-jmt { text | binary }] | [-jmc { true | false }] | [-jsn *`durable-subscriber-name`*]] [-o *`output-file`*] [-d *`output-directory`*] [-v] [[-verbose] | [-quiet]] { *`wsdlurl`* }

Description

wsdl2service creates a new WSDL file containing an HTTP or JMS service definition from an existing WSDL document containing a binding element.

Arguments

The arguments used to manage the WSDL file generation are reviewed in the following table.

Option	Interpretation
`-?`	Displays the online help for this utility.
`-help`	
`-h`	
`-e` *`service-name`*	Specifies the value of the generated `service` element's `name` attribute.
`-p` *`port-name`*	Specifies the value of the generated `port` element's `name` attribute. To specify multiple port elements, separate the names by a space.
`-n` *`binding-name`*	Specifies the binding used to generate the service.
`-jds` {`queue`/`topic`}	Specifies the JMS destination style.

Option	Interpretation
-jpu *jndi-provider-URL*	Specifies the URL of the JMS JNDI provider.
-jcf *initial-context-factory*	Specifies the JMS initial context factory.
-jfn *jndi-connection-factory-name*	Specifies the JMS JNDI connection factory name.
-jdn *jndi-destination-name*	Specifies the JMS JNDI destination name.
-jmt (text/binary)	Specifies the JMS message type.
-jmc (true/false)	Specifies if the `MessageID` is used as the `CorrelationID`.
-jsn *durable-subscriber-name*	Specifies an optional durable subscriber name.
-o *output-file*	Specifies the name of the generated WSDL file.
-d *output-directory*	Specifies the directory in which the generated WSDL is placed.
-v	Displays the version number for the tool.
-verbose	Displays comments during the code generation process.
-quiet	Suppresses comments during the code generation process.
wsdlurl	The path and name of the existing WSDL file.

Using Ant

To call this tool from Ant you execute the `org.apache.cxf.tools.misc.WSDLToService` class.

Example 6 on page 53 shows the **java** task to generate a JMS binding.

Example 6. Generating a JMS Binding From Ant

```
<java classname="org.apache.cxf.tools.misc.WSDLToService"
fork="true">
  <arg value="-transport"/>
  <arg value="jms"/>
  <arg value="-n"/>
  <arg value="JMSSoapBinding"/>
  ...
  <arg value="MyWSDL.wsdl"/>
  <classpath>
    <path refid="fsf.classpath"/>
```

```
  </classpath>
</java>
```

Name

wsdltoservice -transport http/soap — generates a WSDL document containing an Artix ESB C++ Runtime HTTP endpoint

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`*] [-hctr *`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`*

Description

wsdltoservice -transport http/soap adds a Artix ESB C++ Runtime HTTP endpoint to a WSDL document based on the values provided as arguments to the tool.

Required Arguments

The tool has the following required arguments:

Option	Interpretation
wsdlurl	Specifies the WSDL document from which to base the generated WSDL document.

Optional Arguments

The tool has the following optional arguments:

Option	Interpretation
-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 u*seSecureSockets*	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.

Option	Interpretation
-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.

Option	Interpretation
-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.

Option	Interpretation
`-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.

Name

wsdltoservice -transport corba — generates a WSDL document containing an Artix ESB C++ Runtime CORBA endpoint

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`*

Description

wsdltoservice -transport corba adds a Artix ESB C++ Runtime CORBA endpoint to a WSDL document based on the values provided as arguments to the tool.

Required Arguments

The tool has the following required arguments:

Option	Interpretation
`wsdlurl`	The WSDL document from which to base the generated WSDL document.

Optional Arguments

The tool has the following optional arguments:

Option	Interpretation
`-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.

Option	Interpretation
`-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.

Name

wsdltoservice -transport iiop — generates a WSDL document containing an Artix ESB C++ Runtime IIOP tunnel endpoint

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`

Description

wsdltoservice -transport iiop adds a Artix ESB C++ Runtime IIOP tunnel endpoint to a WSDL document based on the values provided as arguments to the tool.

Arguments

The arguments used to manage endpoint generation are reviewed in the following table.

Option	Interpretation
`-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 `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`.

Option	Interpretation
`-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.

Name

wsdltoservice -transport mq — generates a WSDL document containing an Artix ESB C++ Runtime WebSphere MQ endpoint

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`*

Description

wsdltoservice -transport mq adds a Artix ESB C++ Runtime WebSphere MQ endpoint to a WSDL document based on the values provided as arguments to the tool.

Arguments

The arguments used to manage endpoint generation are reviewed in the following table.

Option	Interpretation
`-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.

Option	Interpretation
`-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.

Option	Interpretation
-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 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.

Option	Interpretation
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.
`-d` *`dir`*	Specifies the output directory for the generated contract.
`wsdlurl`	Specifies the name of the WSDL file to process.

Name

wsdltoservice -transport tibrv — generates a WSDL document containing an Artix ESB C++ Runtime Tibco Rendevous endpoint

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`*

Description

wsdltoservice -transport tibrv adds a Artix ESB C++ Runtime Tibco Rendevous endpoint to a WSDL document based on the values provided as arguments to the tool.

Arguments

The arguments used to manage endpoint generation are reviewed in the following table.

Option	Interpretation
`-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.

Option	Interpretation
`-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.

Option	Interpretation
`-d` *`dir`*	Specifies the output directory for the generated contract.
`-L` *`file`*	Specifies the location of your 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.
`wsdlurl`	Specifies the name of the WSDL file to process.

Name

wsdltoservice -transport tuxedo — generates a WSDL document containing an Artix ESB C++ Runtime Tuxedo endpoint

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`*

Description

wsdltoservice -transport tuxedo adds a Artix ESB C++ Runtime Tuxedo endpoint to a WSDL document based on the values provided as arguments to the tool.

Arguments

The arguments used to manage endpoint generation are reviewed in the following table.

Option	Interpretation
-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.

Option	Interpretation
-L *file*	Specifies the location of your 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.
wsdlurl	Specifies the name of the WSDL file to process.

Adding Routes

Artix provides command line tools for adding routes to WSDL documents.

Name

wsdltorouting — adds a route to a WSDL document

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] {*`wsdl`*}

Description

wsdltorouting adds a route to the provided WSDL document. Routes are used by the Artix ESB router to direct messages between endpoints. For more information see Router Guide [../routing/index.htm].

Arguments

The arguments for controlling the generated route are reviewed in the following table.

Option	Interpretation
`-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.
`-d` *`dir`*	Specifies the output directory for the generated contract.
`-o` *`file`*	Specifies the filename for the generated contract.

Option	Interpretation
`-L` *`file`*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.
`-quiet`	Specifies that the tool is to run in quiet mode.
`-versbose`	Specifies that the tool is to run in verbose mode.
`wsdl`	Specifies the name of the WSDL document to which the route is added.

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.

Name

artix validator — validates a WSDL document

Synopsis

`artix validator` [[-?] | [-help] | [-h]] [-s *`schema-url`*...] [-v] [[-verbose] | [-quiet]] {*`wsdlurl`*}

Description

wsdlvalidator validates whether a WSDL document is well-formed and conforms to the WSDL schema.

Arguments

The arguments used to validate WSDL file are reviewed in the following table:

Option	Interpretation
`-?`	Displays the online help for this utility.
`-help`	
`-h`	
`-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.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the validation.
`-quiet`	Suppresses comments during the validation.
`wsdlurl`	The path and name of the existing WSDL file

Using Ant

To execute this tool using Ant set the **java** task's classname property to `org.apache.cxf.tools.validator.WSDLValidator`.

Name

schemavalidator — validates WSDL documents and checks if they meet the WS-I basic profile

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 ]`

Description

schemavalidator validates that a WSDL document is well-formed. In addition, it can test the WSDL document for conformance to the WS-I basic profile.

Arguments

The arguments used to manage WSDL validation are described below.

Argument	Interpretation
`-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.
`-w WSDL_XSD_URL`	Specifies the URL of the document to be validated.
`-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`.

Argument	Interpretation
`-h`	Displays the tool's usage statement.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.

Transforming XML

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

Name

xslttransform — transforms an XML document based on an XSLT stylesheet

Synopsis

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

Description

xslttransform transforms an XML document based on an XSLT stylesheet. The command uses the Artix ESB transformer which is implemented as part of the Artix ESB C++ Runtime. To use it you must source the **artix_env** script located in *`InstallDir`*`cxx_java/bin`.

Arguments

The arguments for controling the transformation are reviewed in the following table.

Option	Interpretation
`-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.
`-PARAM` *`name value`*	Specifies a name/value pair that corresponds to a parameter in the XSLT stylesheet.

Generating Code from WSDL

Artix ESB provides a number of command line tools for generating application code from WSDL documents.

Name

artix wsdlgen — generates application code based on JavaScript templates

Synopsis

`artix wsdlgen` [-G *`ApplicationType`*] [-T *`TemplateID`*...] [-C *`configFile`*] [-D *`name`*=*`value`*...] *`WSDLFile`*

Description

artix wsdlgen is a customizable code generator. Using JavaScript templates, you can customize the implementation classes generated from a WSDL document. The tool includes a number of standard templates that generate basic C++ and Java code if you do not require any customization.

For more information see WSDLGen Guide [../wsdlgen/index.htm].

Arguments

The arguments used to manage the code generation are reviewed in the following table.

Option	Interpretation
`-G` *`ApplicationType`*	Specifies the type of application to generate. The following application types are defined by default: • `cxx`—for generating C++ code • `jaxrpc`—for generating JAX-RPC code • `jaxws`—for generator JAX-WS code
`-T` *`TemplateID`*	Specifies the template ID that governs code generation. See Template IDs on page 87 for details.
`-C` *`ConfigFile`*	Specifies the location of a configuration file to be used by the code generator.
`-D` *`name`*=*`value`*	Specifies the value, *`value`*, of a JavaScript property, *`name`*. Typically you will use this option to specify a value for the portType property. This instructs the code generator the WSDL `portType` element for which code is to be generated.

Option	Interpretation
WSDLFile	Specifies the URL of the WSDL document.

Template IDs

When called with `-G ApplicationType` the `-T TemplateID` switch supports the following template IDs:

Option	Interpretation
`impl`	Generate the stub and skeleton code require to implement the interface defined by the specified WSDL `portType` element.
`server`	Generate a simple `main()` for a standalone service that will host an implementation of the interface defined by the specified WSDL `portType` element. Stub code is also generated.
`client`	Generate a C++ file or Java class that invokes all of the operations defined by the specified WSDL `portType` element. Stub code is also generated.
`plugin`	If generating C++ or JAX-RPC code, generate all of the code needed to implement the interface defined by the specified WSDL `portType` element as an Artix plug-in.
`all`	For C++ and JAX-RPC, generate a client, a server, and an Artix plug-in. For JAX-WS, generate a client and a server. For example, specifying `-G cxx -T all` is equivalent to `-G cxx -T impl -T plugin -T client`.
`ant`	Generate an Apache Ant build file for a Java application.
`make`	Generate a make file for a C++ application.

Name

artix wsdl2cpp — generates C++ stubs and skeletons for the services defined in a WSDL document

Synopsis

`artix wsdl2cpp` [-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] [-user_reserved_words *`word1`* [:*`wordn`*...]] [-L *`file`*] [[-quiet] | [-verbose]] [-h] [-v] *`wsdlurl`*

Description

artix wsdl2cpp generates C++ skeletons for the services defined in a WSDL document. It can also generate starting point code for your server and client applications.

Required Arguments

The tool has the following required arguments:

Option	Interpretation
`wsdlurl`	The WSDL document from which the code is generated.

Optional Arguments

The tool uses the following optional arguments:

Option	Interpretation
`-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_name[: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. This switch can appear multiple times.
`-nexclude URI[=C++namespace]`	Do not generate C++ stub code for the specified XML namespace. You can optionally map the XML namespace to a C++ namespace in case it is referenced by the rest of the XML Schema/WSDL document. This switch can appear multiple times.
`-ninclude URI[=C++namespace]`	Generates C++ stub code for the specified XML namespace. You can optionally map the XML namespace to a 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`.
`-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*—Was needed to support routing in earlier versions.

Option	Interpretation
`-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 *`ServiceNamePortTypeName`*. The service name 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 ESB C++ Runtime 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 ESB C++ Runtime 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`	Dislays detailed information about the options.
`-reflect`	Enables reflection on the generated classes.
`-wrapped`	When used with document/literal wrapped style, generates function signatures with wrapped parameters, instead of unwrapping into separate parameters.

Option	Interpretation
`-user_reserved_words` *word1*`[:`*wordn*`...]`	Specifies a colon-separated list of words to be treated as reserved. For example, **`-user_reserved_words SEC:MILLISEC`** would generate a header file including 'class _SEC' instead of 'class SEC'.
`-L` *file*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-h`	Displays the tool's usage statement.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.

Name

artix wsdl2java — generates JAX-WS compliant Java code from a WSDL document

Synopsis

`artix wsdl2java` [[-?] | [-help] | [-h]] [-fe *frontend*...] [-db *databinding*...] [-wv *wsdlVersion*...] [-p [*wsdlNamespace*=]*PackageName*...] [-b *bindingName*...] [-sn *serviceName*] [-d *output-directory*] [-catalog *catalogName*] [-compile] [-classdir *compile-class-dir*] [-client] [-server] [-impl] [-all] [-ant] [-keep] [-defaultValues[=*DefaultValueProvider*]] [-nexclude *schema-namespace* [=*java-packagename*]...] [-exsh { true | false }] [-dns { true | false }] [-dex { true | false }] [-wsdlLocation *wsdlLocation*] [-xjc*args*] [-noAddressBinding] [-validate] [-v] [[-verbose] | [-quiet]] *wsdlfile*

Description

artix wsdl2java takes a WSDL document and generates fully annotated Java code from which to implement a service. The WSDL document must have a valid `portType` element, but it does not need to contain a `binding` element or a `service` element. Using the optional arguments you can customize the generated code. In addition, **artix wsdl2java** can generate an Ant-based makefile to build your application.

Arguments

The arguments used to manage the code generation process are reviewed in the following table.

Option	Interpretation
`-?`	Displays the online help for this utility.
`-help`	
`-h`	

Option	Interpretation
`-fe` *frontend*	Specifies the front end used by the code generator. The default is `jaxws`.[a]
`-db` *databinding*	Specifies the data binding used by the code generator. The default is `jaxb`.[b]
`-wv` *wsdlVersion*	Specifies the WSDL version expected by the tool. The default is `1.1`.[c]
`-p` [*wsdlNamespace*=]*PackageName*	Specifies zero, or more, package names to use for the generated code. Optionally specifies the WSDL namespace to package name mapping.
`-b` *bindingName*	Specifies zero, or more, JAXWS or JAXB binding files. Use spaces to separate multiple entries.
`-sn` *serviceName*	Specifies the name of the WSDL service for which code is to be generated. The default is to generate code for every service in the WSDL document.
`-d` *output-directory*	Specifies the directory into which the generated code files are written.
`-catalog` *catalogUrl*	Specifies the URL of an XML catalog to use for resolving imported schemas and WSDL documents.
`-compile`	Compiles generated Java files.
`-classdir` *complile-class-dir*	Specifies the directory into which the compiled class files are written.
`-client`	Generates starting point code for a client mainline.
`-server`	Generates starting point code for a server mainline.
`-impl`	Generates starting point code for an implementation object.
`-all`	Generates all starting point code: types, service proxy, service interface, server mainline, client mainline, implementation object, and an Ant `build.xml` file.
`-ant`	Generates the Ant `build.xml` file.
`-keep`	Instructs the tool to not overwrite any existing files.
`-defaultValues`[=*DefaultValueProvider*]	Instructs the tool to generate default values for the generated client and the generated implementation. Optionally, you can also

Option	Interpretation
	supply the name of the class used to generate the default values. By default, the `RandomValueProvider` class is used.
`-nexclude` *`schema-namespace`*`[=`*`java-packagename`*`]`	Ignore the specified WSDL schema namespace when generating code. This option may be specified multiple times. Also, optionally specifies the Java package name used by types described in the excluded namespace(s).
`-exsh (true/false)`	Enables or disables processing of extended soap header message binding. Default is `false`.
`-dns (true/false)`	Enables or disables the loading of the default namespace package name mapping. Default is `true`.
`-dex (true/false)`	Enables or disables the loading of the default excludes namespace mapping. Default is `true`.
`-wsdlLocation` *`wsdlLocation`*	Specifies the value of the `@WebService` annotation's wsdlLocation property.
`-xjc`*`args`*	Specifies a comma separated list of arguments to be passed to directly to the XJC when the JAXB data binding is being used. To get a list of all possible XJC arguments use the `-xjc-X`.
`-noAddressBinding`	Instructs the tool to use the Artix ESB proprietary WS-Addressing type instead of the JAX-WS 2.1 compliant mapping.
`-validate`	Instructs the tool to validate the WSDL document before attempting to generate any code.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.
`wsdlfile`	The path and name of the WSDL file to use in generating the code.

[a]Currently, Artix ESB only provides the JAX-WS front end for the code generator.

[b]Currently, Artix ESB only provides the JAXB data binding for the code generator.

[c]Currently, Artix ESB only provides WSDL 1.1 support for the code generator.

Using Ant

To call the WSDL to Java code generator from Ant set the **java** task's classname property to `org.apache.cxf.tools.wsdlto.WSDLToJava`.

Example 7 on page 95 shows the **java** task to execute this command.

Example 7. Generating a Java Code From Ant

```
<java classname="org.apache.cxf.tools.wsdlto.WSDLToJava"
fork="true">
  <arg value="-client"/>
  ...
  <arg value="MyWSDL.wsdl"/>
  <classpath>
    <path refid="fsf.classpath"/>
  </classpath>
</java>
```

Name

wsdltojava — generates JAX-RPC compliant Java code stubs and skeletons for the services defined in a WSDL document

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] [-merge] [-deployable] [-nexclude *`namespace`* [=*`package`*]] [-ninclude *`namespace`*[=*`package`*]] [-ser] [-stub] [-typehandlers] [-L *`file`*] [[-quiet] | [-verbose]] [-h] [-v] *`wsdlurl`*

Description

wsdltojava generates JAX-RPC compliant Java code stubs and skeletons for the services defined in the specified WSDL document. 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.

Note

The JAX-RPC APIs are implemented as part of the Artix ESB C++ Runtime using a JNI layer.

Required Arguments

The tool has the following required attributes:

Option	Interpretation
`wsdlurl`	Specifies the WSDL document for which the code is generated.

Optional Arguments

The tool has the following optional arguments:

Option	Interpretation
`-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 out_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.

Option	Interpretation
`-deployable`	Generate a deployment descriptor to deploy the generated plug-in into an Artix ESB C++ Runtime container.
`-nexclude` *`namespace`*`[=`*`package`*`]`	Instructs the code generator to skip the specified XML Schema 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 XML Schema 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.
`-typehandlers`	Specifies that the tool will generate C++ style type handlers for complex types. Not all types are supported by the type handler mechanism.
`-L` *`file`*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-h`	Displays the tool's usage statement.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.

Name

artix wsdl2dbservice — generates an Artix ESB Java Runtime intermediary for a database service

Synopsis

`artix wsdl2dbservice` [-jdbctypemappings *`jdbc-type-mapping-file`*] [-db *`data binding name`*...] [-wv *`wsdl version`*...] [-p [*`wsdl namespace`*=]*`Package Name`*...] [-sn [*`wsdl namespace`*=]*`Package Name`*] [-b *`binding-name`*...] [-d *`output-directory`*] [-compile] [-classdir *`compile-classes-directory`*] [-impl] [-server] [-client] [-all] [-ant] [-nexclude *`schema namespace`* [=*`java packagename`*]...] [-exsh (true|false)] [-dns (true|false)] [-dex (true|false)] [-validate] [-wsdlLocation *`wsdlLocation attribute`*] [-v] [[-verbose] | [-quiet]] *`wsdlurl`*

Description

artix wsdl2dbservce takes a WSDL document and generates the code for a Artix ESB Java Runtime database service intermediary.

Required Arguments

The tool has the following required arguments:

Option	Interpretation
`wsdlurl`	Specifies the WSDL document for which the database service is generated.

Optional Arguments

The tool has the following optional arguments:

Option	Interpretation
`-jdbctypemappings` *`jdbc-type-mapping-file`*	Specifies the location of the JDBC to XSD mapping file.
`-db` *`data binding name`*	Specifies the data binding to use. The default is JAXB.

Option	Interpretation
`-wv` *`wsdl version`*	Specifies the WSDL version to use. The default is WSDL 1.1.
`-p` [*`wsdl namespace`*`=`]*`Package Name`*	Specifies the Java package name to use for the generated code. Optionally, you can specify the WSDL namespace mapping to a particular Java package name.
`-sn` [*`wsdl namespace`*`=`]*`Package Name`*	Specifies the service name to use for the generated code. Optionally, you can specify the WSDL namespace.
`-b` *`binding-name`*	Specifies an external JAXWS or JAXB binding files.
`-d` *`output-directory`*	Specifies the directory into which the generated code is placed.
`-compile`	Specifies that the generated code is compiled.
`-classdir` *`compile-classes-directory`*	Specifies the directory into which the compiled class files are placed.
`-impl`	Generates a dummy implementation class.
`-server`	Generates a server mainline for the service.
`-client`	Generates the code needed to deploy a client.
`all`	Generates all starting point code: types, service proxy, service interface, server mainline, client mainline, implementation object, and an Ant `build.xml` file.
`-ant`	Generates an Ant `build.xml`.
`-nexclude` *`schema-namespace`* [`=`*`java-packagename`*]	Ignore the specified WSDL schema namespace when generating code. This option may be specified multiple times. Also, optionally specifies the Java package name used by types described in the excluded namespace(s).
`-exsh` (`true`/`false`)	Enables or disables processing of extended soap header message binding.
`-dns` (`true`/`false`)	Enables or disables the loading of the default namespace package name mapping. Default is `true`.
`-dex` (`true`/`false`)	Enables or disables the loading of the default excludes namespace mapping. Default is `true`.
`-validate`	Enables validating the WSDL before generating the code.
`-v`	Display's the tool's version.

Option	Interpretation
`-quiet`	Specifies that the tool surpresses most messages.
`-verbose`	SPecifies that the tool displays verbose messages.

Name

wsdltodbservice — generates an Artix ESB C++ Runtime intermediary for a database service

Synopsis

`wsdltodbservice` [-d *`dir`*] [-source *`dir`*] [-plugin] [-h] [-v] [[-quiet] | [-verbose]] *`dbconfig wsdlurl`*

Description

wsdltodbservce takes a WSDL document and an Artix ESB C++ Runtime 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.

Required Arguments

The tool has the following required arguments:

Option	Interpretation
`dbconfig`	Specifies the name of the database configuration file to use when generating code.
`wsdlurl`	Specifies the WSDL document to use when generating code.

Optional Arguments

The tool has the following optional arguments:

Option	Interpretation
`-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 ESB C++ Runtime container.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.

Option	Interpretation
`-quiet`	Specifies that the tool is to run in quiet mode.
`-versbose`	Specifies that the tool is to run in verbose mode.

Name

artix java2js — generates JavaScript code from a Java SEI

Synopsis

`artix java2js` [[-?] | [-help] | [-h]] [-jsutils] [-o *`outFile`*] [-d *`outDir`*] [-beans *`beanPath`*...] [-cp *`classpath`*] [-soap12] [-v] [[-verbose] | [-quiet]] *`classname`*

Description

artix java2js takes a compiled Java SEI and generates JavaScript code from which to implement a client that is capable of interacting with a service implementing the service interface.

Arguments

The arguments used to manage the code generation process are reviewed in the following table.

Option	Interpretation
`-?`	Displays the online help for this utility.
`-help`	
`-h`	
`-jsutils`	Instructs the tool to put the Artix ESB JavaScript utility code at the top of the generated file.
`-o` *`outFile`*	Specifies the name of the generated file.
`-d` *`outDir`*	Specifies the name of the directory into which the generated file is placed.
`-beans` *`beanPath`*	Specify the pathname of a file defining additional Spring beans to customize data binding configuration.
`-cp` *`classpath`*	Specifies the classpath used to discover the SEI and required support files.
`-soap12`	Instructs the tool to generate a SOAP 1.2 binding.
`-v`	Displays the version number for the tool.

Option	Interpretation
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.
`classname`	Specifies the name of the SEI class.

Name

artix wsdl2js — generates JavaScript consumer code from a WSDL document

Synopsis

`artix wsdl2js` [[-?] | [-help] | [-h]] [-wv *`wsdlVersion`*] [-p {*`wsdlNamespace`* [=*`jsPrefix`*]}...] [-catalog *`catalogUrl`*] [-d *`outDir`*] [-validate] [-v] [[-verbose] | [-quiet]] *`wsdlUrl`*

Description

artix wsld2js takes a WSDL document and generates JavaScript code from which to implement a consumer capable of interacting with a service provider implementing the described service. The WSDL document must have a valid `portType` element, but it does not need to contain a `binding` element or a `service` element.

Arguments

The arguments used to manage the code generation process are reviewed in the following table.

Option	Interpretation
`-?`	Displays the online help for this utility.
`-help`	
`-h`	
`-wv` *`wsdlVersion`*	Specifies the WSDL version the tool expects. The default is WSDL 1.1. The tool can also use WSDL 1.2.
`-p` *`wsdlNamespace`*[=*`jsPrefix`*]	Specifies a mapping between the namespaces used in the WSDL document and the prefixes used in the generated JavaScript. This argument can be used more than once.
`-catalog` *`catalogUrl`*	Specifies the URL of an XML catalog to use for resolving imported schemas and WSDL documents.
`-d` *`outDir`*	Specifies the directory into which the generated code is written.

Option	Interpretation
`-validate`	Instructs the tool to validate the WSDL document before attempting to generate any code.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.
`wsdlUrl`	Specifies the location of the WSDL document from which the code is generated.

Generating Support Files

Artix provides tools to generate a number of support files.

Name

artix wsdl2corba -idl — generates an IDL file from a WSDL document containing a CORBA binding

Synopsis

`artix wsdl2corba` {-idl} {-b *`binding`*} [-corba] [-i *`portType`*] [-d *`dir`*] [-w *`wsdlOut`*] [-o *`idlOut`*...] [-props *`namespace`*] [-wrapped] [-a *`address`*] [-f *`address-file`*] [[-quiet] | [-verbose]] [-v] [-h] *`wsdl`*

Description

artix wsdl2corba -idl generates an IDL file from a WSDL document containing a CORBA binding. In addition, the tool can be used to add a CORBA binding to a WSDL file and generate an IDL file in one step.

Required Arguments

The tool has the following required arguments:

Option	Interpretation
`-idl`	Specifies that the tool is to generate IDL from the binding.
`-b` *`binding`*	Specifies the name of the CORBA binding for which the IDL file is generated.
`wsdl`	Specifies the WSDL document to which the binding is added.

Optional Arguments

The tool has the following optional arguments:

Option	Interpretation
`-corba`	Specifies that an CORBA binding will be added to the WSDL document. You must also use the `-i` flag in conjunction with this flag.
`-i` *`portType`*	Specifies the name of the port type for which the CORBA binding is generated.
`-d` *`dir`*	Specifies the directory into which the new IDL file is written.

Option	Interpretation
`-w wsdlOut`	Specifies the name of the WSDL document containing the generated CORBA binding.
`-o idlOut`	Specifies the name of the generated IDL file.
`-props namespace`	Specifies the namespace to use for the generated CORBA typemap.
`-wrapped`	Specifies that the generated binding uses wrapped types.
`-a address`	Specifies the value of the generated binding's `corba:address` element's `location` attribute.
`-f address-file`	Specifies the name of a file whose contents are to be used as the value of the generated binding's `corba:address` element's `location` attribute.
`-v`	Displays the tool's version.
`-h`	Specifies that the tool will display a detailed usage statement.
`-quiet`	Specifies that the tool is to run in quiet mode.
`-versbose`	Specifies that the tool is to run in verbose mode.

Name

wsdltocorba -idl — generates an IDL file from a WSDL document containing an Artix ESB C++ Runtime CORBA binding

Synopsis

`wsdltocorba -idl` {-b *`binding`*} [-corba] [-i *`portType`*] [-d *`dir`*] [-o *`file`*] [-L *`file`*] [[-quiet] | [-verbose]] [-h] [-v] *`wsdl`*

Description

wsdltocorba -idl generates an IDL file from a WSDL document containing a Artix ESB C++ Runtime CORBA binding.

Required Arguments

The required arguments for generating an IDL file are reviewed in the following table.

Option	Interpretation
`-b` *`binding`*	Specifies the name of the CORBA binding for which the IDL is generated.
`wsdl`	Specifies the WSDL document to which the binding is added.

Optional Arguments

The optional arguments used to control the generated CORBA binding are explianed in the following table.

Option	Interpretation
`-corba`	Specifies that a CORBA binding is to be generated.
`-i` *`portType`*	Specifies the name of the port type for which the CORBA binding is generated.
`-d` *`dir`*	Specifies the directory into which the new WSDL document is written.
`-o` *`file`*	Specifies the name of the generated WSDL document. The default is *`wsdl_file`*`-corba.wsdl`.
`-props` *`namespace`*	Specifies the namespace to use for the generated CORBA typemap.

Option	Interpretation
`-wrapped`	Specifies that the generated binding uses wrapped types.
`-L` *`file`*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.
`-quiet`	Specifies that the tool is to run in quiet mode.
`-versbose`	Specifies that the tool is to run in verbose mode.

Name

artix sql2dbconfig — generates a Artix ESB Java Runtime database service configuration file

Synopsis

`artix sql2dbconfig` [-d *`output-dir`*] [-new [-driver *`driver-class`*] [-connectionurl *`connection-url`*] [-property *`property`* ...] [-pool] [-maxactive *`pool-maxactive`*] [-maxidle *`pool-maxidle`*] [-transaction *`transaction-level`*] [-autocommit *`auto-commit`*] [-readonly *`read-only`*]] [-test] [-add [-name *`name`*] [-query *`query`*] [-isprocedure] [-isupdate] [-oktoexecute] [-parametertype *`parameter-type`*...] [-parameterdirection *`parameter-direction`*...] [-parameternullable *`parameter-nullable`*...] [-parametervalue *`parameter-value`*...] [-timeout *`timeout`*]] [-delete [-name *`name`*]] [-v] [[-verbose] | [-quiet]] *`outfile`*

Important

Before running this command you should set the `JDBC_DRIVER_CP` environment variable to point to all required JDBC drivers.

Required Arguments

The tool has the following required arguments:

Option	Interpretation
`outfile`	Specifies the path and the name of the generated database service configuration file. The default is `./dbconfig.xml.db`.

Optional Arguments

The the tool has the following optional arguments:

Option	Interpretation
`-d` *`output-directory`*	Specifies the directory into which the generated configuration file is placed.

Option	Interpretation
`-new`	Specifies that a new configuration file is to be generated. Only connection information will be added to the newly created configuration file. Users can add operations to the configuration file in subsequent commands by specifying the `-add` option. The target output file should not exist in the filesystem. The following options can be used with the `new` option: • `-driver` • `-connectionurl` • `-property` • `-pool` • `-maxactive` • `-maxidle` • `-transaction` • `-autocommit` • `-readonly`
`-driver` *`driver-class`*	Specifies the driver class for the new connection. This option can only be used with `-new`.
`-connectionurl` *`connection-url`*	Specifies the connection url for the new connection. This option can only be used with `-new`.
`-property` *`property`*	Specifies a connection property for the new connection. This option can only be used with `-new`.
`-pool`	Specifies that connection pooling should be enabled. This option can only be used with `-new`.
`-maxactive` *`pool-maxactive`*	Specifies the maximum active connections in the connection pool. This option can only be used when both `-new` and `-pool` are specified.

Option	Interpretation
`-maxidle` *`pool-maxidle`*	Specifies the maximum idle connections in the pool. This option can only be used when both `-new` and `-pool` are specified.
`-transaction` *`transaction-level`*	Specifies the transaction isolation level for the new connection. This option can only be used with `-new`. The value should be an integer value as defined in JDBC specification or one of the following: `none`, `read_committed`, `read_uncommitted`, `repeatable_read`, and `serializable`.
`-autocommit` *`auto-commit`*	Specifies the auto commit value for the connection. This option can only be used with `-new`.
`-readonly` *`read-only`*	Specifies the read only value for the connection. This option can only be used with `-new`.
`-test`	Test a connection using the connection information provided in an configuration.
`-add`	Specified when adding a new operation to an existing database service configuration. This option cannot be used with `-new`. The following options can be used with `-add`: • `-name` • `-query` • `-isprocedure` • `-isupdate` • `-oktoexecute` • `-parametertype` • `-parameterdirection` • `-parameternullable` • `-parametervalue` • `-timeout`

Option	Interpretation
`-name` *name*	Specifies the name of the operation.
`-query` *query*	Specifies the query or procedure call of the operation.
`-isprocedure`	Specifies that the operation is a stored procedure.
`-isupdate`	Specifies that the operation will write to the database. If both `-isupdate` and `-isprocedure` are specifies, then `-oktoexecute` must be specified. If the operation requires IN/INOUT parameters, users must provide parameter values for executing the operation. The reason is that a stored procedure can return multiple results. The command will need to execute the stored procedure to obtain the result metadata. However, if only `-isupdate` is specified, the command will not need to execute the operation and the result is assumed to be an update count.
`-oktoexecute`	Specifies that it is OK to execute the operation to get resultset metadata.
`-parametertype` *parameter-type*	Specifies the JDBC type for a parameter by position. Positions start at 1. The value should be the integer or string value of the JDBC type as defined in the JDBC specification. For example `-parametertype 1=3` specifies that the first parameter is a JDBC decimal.
`-parameterdirection` *parameter-direction*	Specifies the direction for a parameter by position. Positions start at 1. The value should be one of `IN`, `INOUT`, and `OUT`. For example, `-parameterdirection 2=IN` specifies that the second parameter is an input parameter.
`-parameternullable` *parameter-nullable*	Specifies whether a parameter is nullable by position. Positions start at 1. For example, `-parameternullable 1=true` specifies that the first parameter can be null.
`-parametervalue` *parameter-value*	Specified a parameter's value for by position. Positions start at 1. This option is necessary for parameterized operation when `-oktoexecute` is used. For example, `-parametervalue 1=12.0` specifies the first parameter's value is `12.0`.
`-timeout` *timeout*	Specifies the number of seconds before the operation times out.

Option	Interpretation
`-delete`	Specifies that an operation is to be deleted from the database service configuration file.
`-name` *name*	Specifies the name of the operation to delete.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the generation.
`-quiet`	Suppresses comments during the generation.

Name

wsdd — generates a deployment descriptor that can be used to deploy a Artix ESB C++ Runtime plug-in into the Artix ESB C++ Runtime container

Synopsis

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

Description

wsdd generates a deployment descriptor that can be used to deploy and Artix ESB C++ Runtime plug-in into the Artix ESB C++ Runtime container.

Required Options

The tool has the following required options:

Option	Interpretation
`-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 ESB C++ Runtime configuration file.
`-pluginType {Cxx | Java}`	Specifies if the plug-in is implemented in C++ or Java.

Optional Arguments

The tool has the following optional arguments:

Option	Interpretation
`-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.

Option	Interpretation
`-file` *file*	Specifies the name of the generated deployment descriptor.
`-d` *dir*	Specifies the directory where the generated file will be written.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.
`-quiet`	Specifies that the tool is to run in quiet mode.
`-versbose`	Specifies that the tool is to run in verbose mode.

Name

artix wsdl2acl — generates a starting point ACL file from a WSDL document

Synopsis

`artix wsdl2acl` {-s *server*} {*WSDL-URL*} [-i *interface*] [-r *default_role*] [-d *output_dir*] [-o *output_file*] [-props *props_file*] [-L *license*] [[-quiet] | [-verbose]] [-v]

Description

artix wsdl2acl generates a starting point ACL file from a WSDL document. The generated ACL must be completed before it can be used.

Required Arguments

The command has the following required arguments:

Option	Interpretation
`-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.

Optional Arguments

The command has the following optional arguments:

Option	Interpretation
`-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.

Option	Interpretation
`-L` *`license`*	Specifies the location of your Artix ESB license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.
`-v`	Displays the tool's version.
`-quiet`	Specifies that the tool is to run in quiet mode.
`-versbose`	Specifies that the tool is to run in verbose mode.