IONA®

Artix™

Command Line Reference

Version 2.1, June 2004

Making Software Work Together™

Updated: 18-Aug-2004

Contents

Generating WSDL 1

Generating from Java Classes 2

Generating from CORBA IDL 4

Generating from a COBOL Copybook 6

Adding Bindings 9

Generating a SOAP Binding 10

Generating a CORBA Binding 11

Validating WSDL 13

Generating Code from WSDL 15

C++ Code Generation 16

Java Code Generation 20

Tools for Generating Support Files 23

Generating IDL from WSDL 24

Generating an ACL File 25

Generating WSDL

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

In this chapter

This chapter discusses the following topics:

Generating from Java Classes	page 2
Generating from CORBA IDL	page 4
Generating from a COBOL Copybook	page 6

Generating from Java Classes

Overview

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

JAVATOWSDL

Synopsis

```
javatowsdl  [-t namespace] [-x namespace] [-i porttype] [-o file]
[-useTypes] [-v] [-?] ClassName
```

Options

The command has the following options:

`-t` *namespace*	Specifies the target namespace of the generated WSDL document. By default, the java package name will be used as the target namespace. If no package name is specified, the generated target namespace will be `http:\\www.iona.com\`*ClassName*.
`-x` *namespace*	Specifies the target namespace of the XMLSchema information generated to represent the data types inside the WSDL document.By default, the generated target namespace of the XMLSchema will be `http:\\www.iona.com\`*ClassName*`\xsd`.
`-i` *porttype*	Specifies the name of the generated `<portType>` in the WSDL document. By default the name of the class from which the WSDL is generated is used.
`-o` *file*	Specifies output file into which the WSDL is written.
`-useTypes`	Specifies that the generated WSDL will use types in the WSDL message parts. By default, messages are generated using wrapped `doc/literal` style. A wrapper element with a sequence will be created to hold method parameters.
`-v`	Prints out the version of the tool.
`-?`	Prints out a help message explaining the command line flags.

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

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

Generating from CORBA IDL

Overview

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

IDLTOWSDL

Synopsis

```
idltowsdl [-useypes][-unwrap][-a address][-f file][-o dir][-s
type][-r file][-L file][-P file][-w namespace][-x namespace][-t
namespace][-T file][-n file][-b] idlfile
```

Options

The command has the following options:

`-usetypes`	Generate rpc style messages. rpc style messages have parts defined using XMLSchema types instead of XML elements.
`-unwrap`	Generate unwrapped doc/literal messages. Unwrapped messages have parts that represent individual elements. Unlike wrapped messages, unwrapped messages can have multiple parts and are not allowed by the WS-I.
`-a` *`address`*	Specifies an absolute address through which the object reference may be accessed. The *`address`* may be a relative or absolute path to a file, or a corbaname URL
`-f` *`file`*	Specifies a file containing a string representation of an object reference. The object reference is placed in the `<corba:address>` element in the `<port>` definition of the generated service. The *`file`* must exist when you run the IDL compiler.
`-o` *`dir`*	Specifies the directory into which the WSDL file is written.
`-s` *`type`*	Specifies the XMLSchema type used to map the IDL sequence`<octet>` type. Valid values are `base64Binary` and `hexBinary`. The default is `base64Binary`.

`-r` *`file`*	Specify the pathname of the schema file imported to define the `Reference` type. If the `-r` option is not given, the idl compiler gets the schema file pathname from `etc/idl.cfg`.
`-L` *`file`*	Specifies that the logical portion of the generated WSDL specification into is written to *`file`*. *`file`* is then imported into the default generated file.
`-P` *`file`*	Specifies that the physical portion of the generated WSDL specification into is written to *`file`*. *`file`* is then imported into the default generated file.
`-w` *`namespace`*	Specifies the namespace to use for the WSDL `targetNamespace`. The default is `http://schemas.iona.com/idl/`*`idl_name`*.
`-x` *`namespace`*	Specifies the namespace to use for the Schema `targetNamespace`. The default is `http://schemas.iona.com/idltypes/`*`idl_name`*.
`-t` *`namespace`*	Specifies the namespace to use for the CORBA TypeMapping `targetNamespace`. The default is `http://schemas.iona.com/typemap/corba/`*`idl_name`*.
`-T` *`file`*	Specifies that the schema types are to be generated into a separate file. The schema file is included in the generated contract using an import statement. This option cannot be used with the `-n` option.
`-n` *`file`*	Specifies that a schema file, `file`, is to be included in the generated contract by an import statement. This option cannot be used with the `-T` option.
`-b`	Specifies that bounded strings are to be treated as unbounded. This eliminates the generation of the special types for the bounded string.

To combine multiple flags in the same command, use a colon delimited list. The colon is only interpreted as a delimiter if it is followed by a dash. Consequently, the colons in a `corbaname` URL are interpreted as part of the URL syntax and not as delimiters.

Note: The command line flag entries are case sensitive even on Windows. Capitalization in your generated WSDL file must match the capitalization used in the prewritten code.

Generating from a COBOL Copybook

Overview

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

COLBOLTOWSDL

Synopsis

`coboltowsdl -b` *`binding`* `-op` *`operation`* `-im [`*`inmessage`*`:]`*`incopybook`* `[-om [`*`outmessage`*`:]`*`outcopybook`*`] [-fm [`*`faultmessage`*`:]`*`faultbook`*`] [-i` *`portType`*`] [-t` *`target`*`] [-x` *`schema_name`*`] [-useTypes] [-o` *`file`*`]`

Parameters

The command has the following required parameters:

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

Options

The command has the following options:

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

`-fm [`*`faultmessage`*`:]`*`faultbook`*	Specifies the name of a fault message and the copybook file from which the data defining the message is taken. The fault message name, *`faultmessage`*, is optional. However, if the copybook has more than one `01` levels, you will be asked to choose the one you want to use as the fault message. You can specify more than one fault message.
`-i `*`portType`*	Specifies the name of the port type in the generated WSDL. Defaults to *`binding`*`PortType`.[a]
`-t `*`target`*	Specifies the target namespace for the generated WSDL. Defaults to `http://www.iona.com/`*`binding`*.
`-x `*`schema_name`*	Specifies the namespace for the schema in the generated WSDL. Defaults to `http://www.iona.com/`*`binding`*`/types`.
`-useTypes`	Specifies that the generated WSDL will use `<types>`. Default is to generate `<element>` for schema types.
`-o `*`file`*	Specifies the name of the generated WSDL file. Defaults to *`binding`*`.wsdl`.

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

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

CHAPTER 2

Adding Bindings

Artix provides a tools for adding bindings to WSDL.

In this chapter

This chapter discusses the following topics:

Generating a SOAP Binding	page 10
Generating a CORBA Binding	page 11

Generating a SOAP Binding

Overview

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

WSDLTOSOAP

Synopsis

`wsdltosoap -i` *`portType`* `-n` *`namespace wsdl_file`* `[-b` *`binding`*`][-d` *`dir`*`] [-o` *`file`*`] [-style {document|rpc}] [-use {literal|encoded}]`

Parameters

The command has the following required parameters:

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

Options

The command has the following options:

`-b` *`binding`*	Specifies the name for the generated SOAP binding. Defaults to *`portTypeBinding`*.
`-d` *`dir`*	Specifies the directory into which the new WSDL file is written.
`-o file`	Specifies the name of the generated WSDL file. Defaults to *`wsdl_file`*`-soap.wsdl`.
`-style`	Specifies the encoding style to use in the SOAP binding. Defaults to `document`.
`-use`	Specifies how the data is encoded. Default is `literal`.

Notes

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

Generating a CORBA Binding

Overview

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

WSDLTOCORBA

Synopsis

`wsdltocorba -corba -i` *`portType`* `[-d` *`dir`*`] [-b` *`binding`*`] [-o` *`file`*`] [-n` *`namespace`*`]` *`wsdl_file`*

Parameters

The command has the following required parameters:

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

Options

The command has the following options:

`-d` *`dir`*	Specifies the directory into which the new WSDL file is written.
`-b` *`binding`*	Specifies the name for the generated CORBA binding. Defaults to *`portType`*`Binding`.
`-o` *`file`*	Specifies the name of the generated WSDL file. Defaults to *`wsdl_file`*`-corba.wsdl`.
`-n` *`namespace`*	Specifies the namespace to use for the generated CORBA typemap

Notes

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

CHAPTER 3

Validating WSDL

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

Overview

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

SCHEMAVALIDATOR

Synopsis

```
schemavalidator [-d schema-directory]* [-wd wsdl-directory] [-s
schema-url]* [-w WSDL_XSD_URL] [-deep] [-wsi] [-wh
wsi-test-tools.home] [-tad BasicProfileAssertions] [-?] [-v]
[-verbose]
```

Parameters

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

Options

You can supply the following optional parameters:

`-d` *`schema-directory`*	Specifies the directory used to search for schemas. This switch can appear multiple times.
`-wd` *`wsdl-directory`*	Specifies the directory to look for the specified contract.
`-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 name of contract to validate.
`-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.
`-?`	Displays detailed information about the tool's options.
`-v`	Displays the version of the tool.
`-verbose`	Displays detailed information on the tools progress as it is validating.
`-verbose`	Send extra diagnostic messages to the console while wsdltocpp is running.

CHAPTER 4

Generating Code from WSDL

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

In this chapter

This chapter discusses the following topics:

C++ Code Generation	page 16
Java Code Generation	page 20

C++ Code Generation

Overview

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

WSDLTOCPP

Synopsis

```
wsdltocpp [options] { WSDL-URL | SCHEMA-URL } [-e web_service_name]
[-t port] [-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}:[exe|lib]] [ -jp plugin_class] [-f] [-server] [-client]
[-sample] [-plugin] [-v] [-license] [-declspec declspec] [-all] [-?]
[-flags] [-upper|-lower|-minimal|-mapper class] [-verbose]
[-reflect]
```

Parameters

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

Options

You can supply the following optional parameters:

`i` *`port_type`*	Specifies the name of the port type for which the tool will generate code. The default is to use the first port type listed in the contract. This switch can appear multiple times.
`-e` *`web_service_name`*	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.
`-t` *`port`*	Specifies the name of the port for which code is generated. The default is to used the first port listed in the contract.
`-b` *`binding_name`*	Specifies the name of the binding to use when generating code. The default is the first binding listed in the contract.

`-d` *`output_dir`*	Specifies the directory to which the generated code is written. The default is the current working directory.
`-n` *`[URI=]C++namespace`*	Maps an XML namespace to a C++ namespace. The C++ stub code generated from the XML namespace, URI, is put into the specified C++ namespace, C++namespace. This switch can appear multiple times.
`-nexclude` *`URI[=C++namespace]`*	Do not generate C++ stub code for the specified XML namespace, URI. You can optionally map the XML namespace, URI, to a C++ namespace, C++namespace, in case it is referenced by the rest of the XML schema/WSDL contract. This switch can appear multiple times.
`-ninclude` *`URI[=C++namespace]`*	Generates C++ stub code for the specified XML namespace, URI. You can optionally map the XML namespace, URI, to a C++ namespace, C++namespace. This switch can appear multiple times.
`-nimport` *`C++namespace`*	Specifies the C++ namespace to use for the code generated from imported schema.
`-impl`	Generates the skeleton code for implementing the server defined by the contract.
`-m {NMAKE | UNIX}:[exe | lib]`	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 `:exe`, or a library, by appending `:lib`. For example, the options, `-impl -m NMAKE:exe`, would generate a Windows makefile to build an executable.
`-f`	Deprecated -- No longer used (was needed to support routing in earlier versions.
`-server`	Generates code for a sample implementation of a server.

`-client`	Generates code for a sample implementation of a client.
`-sample`	Generates code for a sample implementation of a client and a server (equivalent to `-server -client`).
`-plugin`	Generates servant registration code as a Bus plug-in.
`-v`	Displays the version of the tool.
`-license`	Displays the currently available licenses.
`-declspec declspec`	Creates NT 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.
`-?`	Displays help on using the command line tool.
`-flags`	Displays detailed information about the options.
`-verbose`	Send extra diagnostic messages to the console while wsdltocpp is running.
`-reflect`	Enables reflection on generated data classes.
`-wrapped`	When used with document/literal wrapped style, generates function signatures with wrapped parameters, instead of unwrapping into separate parameters.

Generated files

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

The following stub files are generated:

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

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

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

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

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

Java Code Generation

Overview

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

WSDLTOJAVA

Synopsis

```
wsdltojava [-e service][-t port][-b binding][-i portType][-d
output_dir][-p [namespace=]package][-impl][-server][-client]
[-types][-interface][-sample][-all][-ant][-datahandlers]
[-nexclude namespace[=package]] [-ninclude namespace[=package]]
artix-contract
```

Description

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

Options

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

`-e` *`service`*	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.
`-t` *`port`*	Specifies the name of the port for which code is generated. The default is to use the first port listed in the service.
`-b` *`binding`*	Specifies the name of the binding to use when generating code. The default is to use the first binding listed in the contract.
`-i` *`portType`*	Specifies the name of a portType for which code will be generated. You can specify this flag for each portType for which you want code generated. The default is to use the first portType in the contract.

`-d` *`output_dir`*	Specifies the directory to which the generated code is written. The default is the current working directory.
`-p` *`[namespace=]package`*	Specifies the name of the Java package to use for the generated code. You can optionally map a WSDL namespace to a particular package name if your contract has more than one namespace.
`-impl`	Generates the skeleton class for implementing the server defined by the contract.
`-server`	Generates a simple main class for the server.
`-client`	Generates only the Java interface and code needed to implement the complex types defined by the contract. This flag is equivalent to specifying `-interface -types`.
`-types`	Generates the code to implement the complex types defined by the contract.
`-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.
`-nexclude` *`namespace[=package]`*	Instructs the code generator to skip the specified XMLSchema namespace when generating code. You can optionally specify a package name to use for the types that are not generated.
`-ninclude` *`namespace[=package]`*	Instructs the code generator to generate code for the specified XMLSchema namespace. You can optionally specify a package name to use for the types in the specified namespace.

Generated files

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

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

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

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

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

Generated type packages

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

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

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

Exceptions

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

CHAPTER 5

Tools for Generating Support Files

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

In this chapter

This chapter discusses the following topics:

Generating IDL from WSDL	page 24
Generating an ACL File	page 25

Generating IDL from WSDL

Overview

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

WSDLTOCORBA

Synopsis

`wsdltocorba -idl -b` *`binding`* `[-corba] [-i` *`portType`*`] [-d` *`dir`*`] [-o` *`file`*`]` *`wsdl_file`*

Parameters

The command has the following required parameters:

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

Options

The command has the following options:

`-corba`	Instructs the tool to generate a CORBA binding for the specified port type.
`-i` *`portType`*	Specifies the name of the port type being mapped to a CORBA binding.
`-d` *`dir`*	Specifies the directory into which the new WSDL file is written.
`-o` *`file`*	Specifies the name of the generated WSDL file. Defaults to *`wsdl_file`*`.idl`.

Notes

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

Generating an ACL File

Overview

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

WSDLTOACL

Synopsis

`wsdltoacl -s` *`server WSDL-URL`* `[-i` *`interface`*`] [-r` *`default_role`*`] [-d` *`output_dir`*`] [-o` *`output_file`*`] [-props` *`props_file`*`] [-v] [-?]`

Parameters

The command has the following required parameters:

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

Options

The command has the following options:

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