Oracle® Service Architecture Leveraging Tuxedo (SALT)

Reference Guide

10*g* Release 3 (10.3)

January 2009

ORACLE

Service Architecture Leveraging Tuxedo Reference Guide, 10*g* Release 3 (10.3)

Oracle SALT Command Reference

The Oracle SALT Command Reference describes system processes and commands delivered with the Oracle SALT software.

Table 1 lists Oracle SALT commands and functions.

Table 1 Oracle SALT Commands and Functions

Name	Description
`buildscaclient`	Builds processes that call SCA components.
`buildscacomponent`	Builds SCA components.
`buildscaserver`	Parses SCDL definitions and interfaces and produces a Tuxedo-deployable server and elements.
`GWWS(5)`	Web service gateway server.
`mkfldfromschema`, `mkfld32fromschema`	The `mkfldfromschema` and `mkfld32fromschema` commands take an XML schema as input and produce a field table.
`mkviewfromschema`, `mkview32fromschema`	The `mkviewfromschema` and `mkview32fromschema` commands take an XML schema as input and produce a view file.
`scaadmin`	SCA servers management command interpreter

Table 1 Oracle SALT Commands and Functions

Name	Description
scapasswordtool	Manages passwords for Tuxedo authentication in SCA clients
setSCAPasswordCallback(3c)	Sets the callback for retrieving a password associated with an identifier in a <binding.atmi> element.
tmscd(1)	Command line utility to activate and deactivate service contract discovery
tmwsdlgen	WSDL document generator.
tuxscagen	Generates SCA, SCDL, and server side interface files for Tuxedo services.
wsadmin	Oracle SALT administration command interpreter.
wsdlcvt	WSDL document converter.
wsloadcf	Reads SALT Deployment file and other referenced artifacts. Loads a binary SALTCONFIG file.

buildscaclient

Name

buildscaclient – Builds processes that call SCA components.

Synopsis

```
buildscaclient -c defaultcomponent[-v] [-k] [-o name]
[-s scaroot][-f firstfiles] [-l lastfiles]
```

Description

This command is used to build client processes that can call SCA components hosted in Tuxedo environments. The command combines files, specified using the -f and -l options, with the SCA and standard Tuxedo ATMI libraries to form a client application. The client application is built using the default C++ language compile command defined for the operating system in use, unless overridden using the CC environment variable.

All specified .c and .cpp files are compiled in one invocation of the compilation system based on the operating system. Users may specify the compiler to invoke by setting the CC environment variable to the name of the compiler. If the CC environment variable is not defined when buildscaclient is invoked, the default C++ language compile command for the operating system is invoked to compile all .c and .cpp files.

You may specify additional options to be passed to the compiler by setting the CFLAGS or the CPPFLAGS environment variables. If CFLAGS is not defined when buildobjclient is invoked, then buildscaclient uses the value of CPPFLAGS, if that variable is defined.

Parameters and Options

buildscaclient supports the following parameters and options:

-c defaultcomponent
Required parameter. Indicates which component should be used for this application.

[-v]
Specifies that the buildscaclient command should work in verbose mode. In particular, it writes the compile command to its standard output.

[-k]
Maintains the generated stubs. buildscaclient generates proxy files that allow dynamic interfacing of clients and references. This is normally compiled and then removed when the proxy is built. This option indicates that the source file should be retained.

Caution: The generated contents of this file may change from release to release. It is advised that you do not depend on the data structures and interfaces exposed in this file. This option is provided to aid in debugging of build problems.

[-o name]
Specifies the name of the client application generated by this command. If the name is not supplied, the application file is named client<.type>, where type is an extension that is dependent on the operating system. For example, on a UNIX system, there would not be a type, but on a Windows system, the type would be .EXE.

[-s scaroot]
Specifies the location of SCA root, where the SCDL files for the required components are located. If not set, the APPDIR environment value is used.

[-f firstfiles]
Specifies the file to be included first in the compile and link phases of the buildscaclient command. The specified file is included before the SCA libraries are included. There are two ways of specifying a file or files:

Filename Specification	Description
`-f firstfile`	One file is specified
`-f "file1.cpp file2.cpp file3.cpp …"`	Multiple files may be specified if their names are enclosed in quotation marks and are separated using white spaces.

Note:

- Filenames that include spaces are not supported.
- The `-f` option may be specified multiple times.

`[-l lastfiles]`

Specifies a file to be included last in the compile and link phases of the `buildscaclient` command. The specified file is included after the SCA libraries are included. There are two ways of specifying the file, as shown in the following table.

Filename Specification	Description
`-l lastfile`	One file is specified
`-l "file1.cpp file2.cpp file3.cpp …"`	Multiple files may be specified if their names are enclosed in quotation marks and are separated using white spaces.

Note:

- Filenames that include spaces are not supported.
- The -l option may be specified multiple times.

Environment Variables

Following is a list of environment variables for `buildSCAclient`:

`TUXDIR`

Finds the SCA libraries and includes files to use when compiling the client applications.

CC

Indicates the compiler for all files with `.c` or `.cpp` file extensions. If not defined, the default C++ language compile command is invoked to compile all `.c` and `.cpp` files, based on the operating system.

CFLAGS

Indicates any arguments that are passed as part of the compiler command line for any files with `.c` or `.cpp` file extensions. If `CFLAGS` does not exist in the `buildscaclient` command environment, the command checks for the `CPPFLAGS` environment variable.

Note: Arguments passed by the `CFLAGS` environment variable take priority over the `CPPFLAGS` variable.

CPPFLAGS

Contains a set of arguments that are passed as part of the compiler command line for any files with `.c` or `.cpp` file extensions.

This is in addition to the command line option `"-I$(TUXDIR)/include"` for UNIX systems or the command line option `/I%TUXDIR%\include` for Windows systems, which is passed automatically by the `buildscaclient` command. If `CPPFLAGS` does not exist in the `buildscaclient` command environment, no compiler commands are added.

LD_LIBRARY_PATH (UNIX systems)

Indicates the directories that contain shared objects to be used by the compiler, in addition to the objects shared by the CORBA software. A colon (:) is used to separate the list of directories. Some UNIX systems require different environment variables:

- HP-UX systems use the SHLIB_PATH environment variable
- AIX systems use LIBPATH

LIB (Windows systems)

Indicates a list of directories that contain the library files. A semicolon (;) is used to separate the list of directories.

Portability

This command is available on any platform on which the Oracle SALT environment is supported.

Example

```
buildscaclient -s /myApplication/scaSrc/uBike
-c SearchRedBikes
```

See Also

`buildscaserver(1), buildscacomponent(1)`

buildscacomponent

Name

`buildscacomponent` - builds SCA components

Synopsis

```
buildscacomponent [-v] [-s scaroot] [-f firstfiles] [-l lastfiles] -c
compositename[/componentname][,compositename,..]] [-y] [-k] [-h]
```

Description

`buildscacomponent` is used to build individual SCA components from source code. The command reads SCDL source, finds the component(s) in the composite(s) file(s) specified, parses the corresponding .componentType file(s) and produces corresponding executable libraries, in the same location as the .componentType files.

The command automatically builds component implementations based on the contents of `<implementation.cpp>` elements as follows:

- The value of `/implementation.cpp/@header` is used to determine the name of the source and `componentType` files containing the implementation.

  For example, an element such as

  ```
  <implementation.cpp library="myLib" header="myComponentImpl.h"/>
  ```

  causes `buildscacomponent` to look for a myComponentImpl.cpp file and compile it, along with stubs generated from its interface located in a corresponding myComponentImpl.componentType file.

Composites may contain one or more components, and the `buildscacomponent` command may build one or more composites in one pass. If more than one component is built, the files specified using the `-f` and `-l` switches are included in each component. To build a single component, the `-c composite/component` syntax should be used. This addresses the cases where individual components are made up of specific sets of source code or libraries.

All specified `.c` and `.cpp` files are compiled in one invocation of the compilation system for the operating system in use. Users may specify the compiler to be invoked by setting the `CC` environment variable to the name of the compiler. If the `CC` environment variable is not defined when `buildscacomponent` is invoked, the default C++ language compile command for the operating system in use is invoked to compile all `.c` and `.cpp` files.

Users may specify options to be passed to the compiler by setting the `CFLAGS` or the `CPPFLAGS` environment variable. `If CFLAGS` is not defined but `CPPFLAGS` is defined when `buildscacomponent` is invoked, the `CPPFLAGS` value is used.

Parameters and Options

`buildscacomponent`supports the following parameters and options:

`[-v]`
Specifies that `buildscacomponent` should work in verbose mode.

`[-s scaroot]`
Specifies the location of the SCA root, where the SCDL file(s) for the component(s) is (are) located, and where the source code of components is processed.

If not specified, the value of APPDIR is used.

`[-f firstfiles]`
Specifies a file to be included first in the compile and link phases of the `buildscacomponent` command. The specified file is included before the SCA libraries are included. There are two ways of specifying a file or files, as shown in Table 2.

Table 2 File Specification Using [-f firstfiles]

Filename Specification	Definition
`-f firstfile`	One file is specified
`-f "file1.cpp file2.cpp file3.cpp …"`	Multiple files may be specified if their names are enclosed in quotation marks and are separated by white space.

Note: Filenames that include spaces are not supported.
The `-f` option may be specified multiple times.

`[-l lastfiles]`
Specifies a file to be included last in the compile and link phases of the `buildscacomponent` command. The specified file is included after the SCA libraries are included. There are two ways of specifying a file, as shown in Table 3.

Table 3 File Specification Using [-l lastfiles]

Filename Specification	Definition
`-l lastfile`	One file is specified
`-l "file1.cpp file2.cpp file3.cpp …“`	Multiple files may be specified if their names are enclosed in quotation marks and are separated by white space.

Note: Filenames that include spaces are not supported.
The `-l` option may be specified multiple times.

`-c {composite[,composite]|composite/component}`

Specifies the name(s) of the composite(s) processed. The composite(s) is (are) searched in `APPDIR` or in the SCDL directory specified above with the -s switch. If it cannot be found, the component libraries are not built.

A list of composites may be specified, in which case all the components listed in the composites will be built. If any of the composites cannot be found or an error is detected (incorrect name, composite does not have any ATMI service binding), a warning message is displayed and the user is prompted to confirm whether the command should continue processing or abort.
If the composite/component notation is used, a single component contained in the specified composite is allowed. This notation covers the situation where specific source files specified with `-f` and `-l` need to be included in the build process of a component.

`[-y]`

Optionally forces processing of input files, automatically ignoring warnings, such as composites specified using the `-c` switch but not physically present from the root directory.

`[-k]`

Keeps the generated proxy and wrapper source. `buildscacomponent` generates proxy and wrapper code with data structures such as the method operation and parameter handling. This is normally compiled and then removed when the component is built. This option indicates that the source file should be kept (to see what the source filename is, use the `-v` option).

Note: The generated contents of this file may change from release to release; DO NOT count on the data structures and interfaces exposed in this file. This option is provided to aid in debugging of build problems.

Environment Variables

TUXDIR

Finds the SCA libraries and include files to use when compiling the client applications.

APPDIR

Indicates the SCA application root location, where the top-level composite should reside.

CC

Indicates the compiler to use to compile all files with `.c` or `.cpp` file extensions. If not defined, the default C++ language compile command for the operating system in use will be invoked to compile all `.c` and `.cpp` files.

CFLAGS

Indicates any arguments that are passed as part of the compiler command line for any files with a `.c` or `.cpp` file extensions. If `CFLAGS` does not exist in the `buildscacomponent` command environment, the `buildscacomponent` command checks for the `CPPFLAGS` environment variable.

CPPFLAGS

Note: Arguments passed by the `CFLAGS` environment variable take priority over the CPPFLAGS variable.

Contains a set of arguments that are passed as part of the compiler command line for any files with a `.c` or `.cpp` file extensions.

This is in addition to the command line option `-I$(TUXDIR)/include` for UNIX systems or the command line option `/I%TUXDIR%\include` for Windows systems, which is passed automatically by the `buildscacomponent` command. If `CPPFLAGS` does not exist in the `buildscacomponent` command environment, no compiler commands are added.

LD_LIBRARY_PATH (UNIX systems)

Indicates which directories contain shared objects to be used by the compiler, in addition to the objects shared by the CORBA software. A colon (:) is used to separate the list of directories. Some UNIX systems require different environment variables: for HP-UX systems, use the `SHLIB_PATH` environment variable; for AIX, use `LIBPATH`.

LIB (Windows systems)

Indicates a list of directories within which to find libraries. A semicolon (;) is used to separate the list of directories.

Portability

`buildscacomponent` is supported on any platform on which the Oracle SALT environment is supported.

Examples

```
buildscacomponent -f utils.c -c searchInventory,updateItem
```

See also

`buildscaserver`

buildscaserver

Name

`buildscaserver` – Builds a Tuxedo server containing SCA component.

Synopsis

```
-o servername -c composite[,composite][-v][-s scaroot]
[-w] [-r rmname][-y] [-k] [-t]
```

Description

`buildscaserver` is used to build a Tuxedo server that is used to route requests to SCA components previously built with the buildscacomponent command. The command generates a main routine that contains bootstrap routines to route Tuxedo or SCA requests to SCA components, and compiles it to form a server host application. The server host application is built using the default C++ compiler provided for the platform.

If the scdl code contains references or services with `<binding.ws>` elements, these will automatically be converted into WSDF files for use by the Web Services gateway (GWWS). All SCA servers built using `buildscaserver` are multi-threaded servers.

Parameters and Options

`buildscaserver` supports the following parameters and options:

`-o servername`
Required. Specifies the name of the server application generated by this command.

`-c compositename[,compositename]`
Required. Specifies the name of the composite hosted. The composite is searched for starting in `APPDIR`, or in the SCDL directory specified above with the `-s` switch. If it is not found, the server is not built. In case you specify a list of composites, then all the listed composites are hosted by the same Tuxedo server.

If any of the composites are not found or an error is detected such as `incorrect name` or `composite does not have any atmi service binding`, a warning message is

displayed and the user is prompted to confirm whether the command should continue processing or abort.

[-v]

Specifies that `buildscaserver` should work in verbose mode.

[-s scaroot]

Specifies the target location of the SCA root, where the SCDL files for the components to be deployed are located.

This directory has a layout suitable to SCA composites and components. Each composite is represented as a directory and contains components in the run-time form, which includes SCDL code and libraries. At run time, the server application uses this directory to find the run-time SCA components.

If components are using the Web Services binding, the root location also receives a WSDF definition file.

[-w]

Specifies that the generated server will host WebServices binding enabled components. By default, a server hosting ATMI binding enabled components is generated. Both types of servers can host the same actual components simultaneously (i.e. there can exist an ATMI and a WS servers, both hosting the same components previously built using the `buildscacomponent` command).

[-r rmname]

Specifies the resource manager associated with this server. The value `rmname` must appear in the resource manager table located in `$TUXDIR/udataobj/RM` on UNIX systems or `%TUXDIR%\udataobj\RM` on Windows systems. Each entry in this file is of the following form:

```
rmname:rmstructure_name:library_names
```

Using the `rmname` value, the entry in `$TUXDIR/udataobj/RM` or `%TUXDIR%\udataobj\RM` automatically includes the associated libraries for the resource manager and sets up the interface between the transaction manager and the resource manager. The value `TUXEDO/SQL` includes the libraries for the Oracle Tuxedo System/SQL resource manager. Other values can be specified once they are added to the resource manager table. If the `-r` option is not specified, the null resource manager is used, by default.

[-y]

Optionally forces processing of input files, automatically ignoring warnings.

[-k]

Keeps the server main stub. `buildscaserver` generates a main stub with data structures such as the service table and a `main()` function. This is normally compiled and then removed when the server is built. This option indicates that the source file should be retained.

Note: To see the source filename, use the `-v` option.

Caution: The generated contents of this file may change from release to release. It is advised that you do not depend on the data structures and interfaces exposed in this file. This option is provided to aid in debugging build problems.

[-t]

Not used in current release.

Environment Variables

TUXDIR

Finds the SCA libraries and include files to use when compiling the client applications.

CC

Indicates the compiler to use to compile all files with `.c` or `.cpp` file extensions. If not defined, the default C++ language compile command is invoked to compile all `.c` and `.cpp` files.

CFLAGS

Indicates any arguments that are passed as part of the compiler command line for any files with a `.c` or `.cpp` file extensions. If `CFLAGS` does not exist in the `buildscaserver` command environment, the `buildscaserver` command checks for the `CPPFLAGS` environment variable.

Note: Arguments passed by the `CFLAGS` environment variable take priority over the `CPPFLAGS` variable.

CPPFLAGS

Contains a set of arguments that are passed as part of the compiler command line for any files with a `.c` or `.cpp` file extensions.

This is in addition to the command line option `"-I$(TUXDIR)/include"` for UNIX systems or the command line option `/I%TUXDIR%\include` for Windows systems, which is passed automatically by the `buildscaserver` command. If CPPFLAGS does not exist in the `buildscaserver` command environment, no compiler commands are added.

LD_LIBRARY_PATH (UNIX systems)
Indicates the directories that contain shared objects to be used by the compiler, in addition to the objects shared by the CORBA software. A colon (`:`) is used to separate the list of directories. Some UNIX systems require different environment variables:

- HP-UX systems use `SHLIB_PATH`
- AIX systems use `LIBPATH`

LIB (Windows only)
Indicates a list of directories where libraries are available. A semicolon (`;`) is used to separate the list of directories.

Portability

This command is available on any platform on which the Oracle SALT server environment is supported.

Examples

```
buildscaserver -o uBikeServer -f utils.c
-d /myApplication/myModules/uBike
-s /myApplication/uBikeComponents/uBike
-c searchInventory,updateItem
```

Error Reporting

This command checks for the following inconsistencies in the SCDL code and reports error messages if:

- at least one syntax error in the SCDL files
- none of the composites contain any service with an ATMI binding
- at least one composite contains services defining ATMI bindings with incompatible `<remoteAccess>` elements. `<remoteAccess>` elements with a value of WorkStation are not supported by this command.
- `/binding.atmi/@requires` contains a legacy value and `/binding.atmi/map` elements contain values that conflict (for example, the same Tuxedo service name mapped to two or more different methods)

See Also

GWWS(5), buildscacomponents(1)

GWWS(5)

Name

GWWS – Web service gateway server.

Synopsis

```
GWWS SRVGRP="identifier" SRVID=number [other_parms]
CLOPT="-A -- –i InstanceID"
```

Description

The GWWS server is the Web service gateway for Tuxedo applications, the core component of Oracle SALT. The GWWS gateway server provides communication with Web service programs via SOAP 1.1/1.2 protocols. The GWWS server has bi-directional (inbound/outbound) capability. It can accept SOAP requests from Web service applications and passes Tuxedo native calls to Tuxedo services (inbound). It also accepts Tuxedo ATMI requests and passes SOAP calls to Web service applications (outbound). GWWS servers are used as Tuxedo system processes and are described in the *SERVERS section of the UBBCONFIG file.

The CLOPT option is a string of command-line options passed to the GWWS server when it is booted. The GWWS server accepts the following CLOPT options:

`-i InstanceID`
: Specifies the GWWS instance unique ID. It is used to distinguish multiple GWWS instances provided in the same Tuxedo domain. This value *must* be unique among multiple GWWS items within the UBBCONFIG file.

Note: The InstanceID value must be pre-defined in the <WSGateway> section of the Oracle SALT Deployment File.

Environment Variables

The SALTCONFIG environment variable must be set before the GWWS server is booted. Accesslog(5) can be enabled by setting environment variable TMENABLEALOG=y .

Deprecation

The following SALT 1.1 GWWS parameter is deprecated in the current release.

`-c Config_file`
: Specifies the SALT 1.1 configuration file.

Note: Starting with the SALT 2.0 release, the GWWS server loads the SALT configuration from the binary SALTCONFIG file instead of the XML-based configuration file. The

configuration file is no longer a GWWS server input parameter. The SALTCONFIG file must be generated using wsloadcf before booting GWWS servers.

Diagnostics

For inbound call, if an error occurs during SOAP message processing, the error is logged. The error is also translated into appropriate SOAP fault and/or HTTP error status code and returned to the Web service client.

For outbound call, if an error occurs during processing, the error is logged. The error is also translated into appropriate Tuxedo system error code (tperrno) and returned to the Tuxedo client.

Examples

Listing 1 GWWS Description in the UBBCONFIG File

```
*SERVERS

GWWS SRVGRP=GROUP1 SRVID=10
       CLOPT="-A -- –i GW1"
GWWS SRVGRP=GROUP1 SRVID=11
       CLOPT="-A -- –i GW2"
GWWS SRVGRP=GROUP2 SRVID=20
       CLOPT="-A -- –i GW3"
```

See Also

UBBCONFIG(5)

tmwsdlgen

SALT Deployment File Reference

SALT Web Service Definition File Reference

mkfldfromschema, mkfld32fromschema

The mkfldfromschema and mkfld32fromschema commands take an XML schema as input and produce a field table. This table can be processed by the mkfldhdr or mkfldhdr32 command or

is loaded by programs that need it. mkfldfromschema is used with 16-bit FML and mlfld32fromschema is used with 32-bit FML.

These commands have the following restrictions:

- Attributes cannot be specified
- Restrictions are ignored because their meaning is application-related

Name

mkfldfromschema, mkfld32fromschema – Generates field table from an XML schema

Synopsis

mkfldfromschema {-i schema|-u schemaurl} -b basenumber -o outputfile**mkfld32fromschema** {-i schema|-u schemaurl} -b basenumber -o outputfile

Description

These commands take an XML schema as input and generate a field table. The XML schema may be specified using either the -i option or the -u option. If neither option is specified, the schema is read from standard input.

Parameters and Options

mkfldfromschema and mkfld32fromschema supports the following options:

-b basenumber
Adds a *base basenumber line to the generated field table.

-i schema
Displays the name of a file containing an XML schema. The -i option cannot be specified in conjunction with the -u option.

-u schemaurl
A URL where the input schema is located. The URL must start with http://. The -u option cannot be specified in conjunction with the -i option.

-o outputfile
The name of a file that will contain the field table. If this option is not specified, the field table will be written to standard output.

Portability

These commands are available on any platform that supports the Oracle Tuxedo server environment.

See Also

`mkviewfromschema, mkview32fromschema`

mkviewfromschema, mkview32fromschema

The `mkviewfromschema` and `mkview32fromschema` commands take an XML schema as input and produce a view file. This file can be processed by the `viewc` or `viewc32` command. `mkviewfromschema` is used with 16-bit views and `mkview32fromschema` is used with 32-bit views.

Name

`mkviewfromschema, mkview32fromschema` – Generates view table from an XML schema

Synopsis

```
mkviewfromschema {-i schema|-u schemaurl} -o outputfile
```

```
mkview32fromschema {-i schema|-u schemaurl} -o outputfile
```

Description

These commands take an XML schema as input and generate a view file. The XML schema may be specified using either the `-i` option or the `-u` option. If neither option is specified, the schema is read from standard input.

Options

`mkviewfromschema, mkview32fromschema` supports the following options:

`-i schema`
The name of a file containing an XML schema. The `-i` option cannot be specified in conjunction with the `-u` option.

`-u schemaurl`
A URL where the input schema is located. The URL must start with http://. The `-u` option cannot be specified in conjunction with the -i option.

`-o outputfile`
The name of a file that contains the output view file. If this option is not specified, the field table is written to standard output.

Portability

These commands are available on any platform that supports the Oracle Tuxedo server environment.

See Also

`mkfldfromschema, mkfld32fromschema`

SDO for C++ Specification V2.1published December, 2006

scaadmin

Name

`scaadmin` – SCA server management command interpreter

Synopsis

```
scaadmin [-v]
```

Description

Use the `scaadmin` command to dynamically redeploy SCA composites or display statistics and status of individual services. The `TUXCONFIG` environment variable is used to determine the location where the Tuxedo configuration file is loaded.

This command has no effect on servers that have not been built using the `buildscaserver(1)` command.

Options

The `scaadmin` command supports the following option:

`[-v ]`
: Causes `scaadmin` to display the Oracle SALT version number, SALT Patch Level. The command exits after print out.

  `scaadmin` must run on an active node.

Commands

`default [-m machine] [-g groupename] [-i srvid] ] [-s servername]`
: Sets the corresponding argument to be the default machine name, groupname, server id, or servername. If the default command is entered with no arguments, the current defaults are printed.

reload [-m machine] [-g groupname] [-i srvid]] [-s servername]
This command dynamically reloads the SCA components hosted on Tuxedo servers. The `-m`, `-g`, `-i` and `-s` options can be used to restrict the reloaded servers to any combination of machine, group, server id and server name.

printstats [-m machine] [-g groupname] [-i srvid] [-s servername]
This command displays the list of services hosted by a server and the associated method, number of queries, and status (`active`, `idle`). The `-m`, `-g`, `-i` and `-s` options can be used to restrict the reloaded servers to any combination of machine, group, server id and server name.

verbose (v) [{off | on}]
Produces output in verbose mode. If no option is given, the current setting is toggled and the new setting is printed. The initial setting is set to `off`.

help (h) [{command | all}]
Prints help messages. If command is specified, the abbreviation, arguments, and description for that command are printed. all causes a description of all commands to be displayed. Omitting all arguments causes the syntax of all commands to be displayed.

echo (e) [{off | on}]
Echoes input command lines when set to on. If no option is given, the current setting is toggled, and the new setting is printed. The initial setting is `off`.

quit (q)
Terminates the session

Interoperability

The `scaadmin` command must run on an active node.

Environment Variables

TUXCONFIG
Used to determine the location where the Tuxedo configuration file is loaded.

Portability

`scaadmin` is supported on any platform that supports the Oracle SALT server environment.

Examples

The following command reloads all the composites hosted by the `uBikeServer` Tuxedo application server, which was built using the `buildscaserver(1)` command.

```
scaadmin
> reload -s uBikeServer
```

The following command displays statistics on the services offered by the `uBikerServer` Tuxedo application server, which was built using the `buildscaserver(1)` command.

```
scaadmin

> printstats -s uBikeServer

Service            Method             Status      Requests
                                                  Processed

SEARCHINVENTORY    searchInventory    A           37
```

scapasswordtool

Name

`scapasswordtool` – Manages passwords for Tuxedo authentication in SCA clients.

Synopsis

```
scapasswordstore -i passwordidentifier -[a|d]
```

Description

This command manages the `password.store` file used by SCA components to refer to Tuxedo-based services.

Passwords are prompted and encrypted. The encrypted version is stored in this file, associated with a clear-text identifier. This command is also used to delete identifier/password pairs from the file.

The password is limited to 40 characters. If standard input is not a terminal, that is, if the user cannot be prompted for a password (as with a Here file, for example), then the `APP_PW` environment variable is accessed to set the password. If the `APP_PW` environment variable is not set and standard input is not a terminal, then `scapasswordtool` prints an error message and exits.

A `password.store` file is created in the current directory if it does not previously exist.

Parameters and Options

`-i passwordidentifier`

Required. The identifier specified in the `<binding>` element. SCA components search the password for this element.

-[a|d]

The -a option adds an identifier/password pair, whereas the -d option deletes it. An error message is printed out and the command processing is aborted in one of the following situations:

- If -a is used to add an already existing identifier
- If -d is used to delete a non-existing identifier

Portability

This command is available on any platform that supports the Oracle Tuxedo server environment.

See Also

setSCAPasswordCallback(3c)

setSCAPasswordCallback(3c)

Name

setSCAPasswordCallback() – Sets the callback for retrieving a password associated with an identifier in a <binding.atmi> element.

Synopsis

```
#include <tuxsca.h>
void setSCAPasswordCallback(char * (_TMDLLENTRY *)(*disp) (char
*identifier))
```

Description

setSCAPasswordCallback() allows an SCA component to identify the callback that returns the clear-text password that is passed to the appropriate authentication code.

The function pointer passed on the call to setSCAPasswordCallback() must conform to the specified parameter definition. The _TMDLLENTRY macro is required for Windows-based operating systems to obtain the proper calling conventions between the Tuxedo libraries and your code. On UNIX systems, the _TMDLLENTRY macro is not required because it expands to the null string.

The identifier points to the password identifier passed to the callback function. The callback function then returns a char * that points to the actual clear-text password.

Return Values

The `setSCAPasswordCallback()` function does not return any data.

Errors

On failure, `setSCAPasswordCallback()` sets `tperrno` to one of the following values:

`[TPEPROTO]`
`setSCAPasswordCallback()` has been called in an improper context.

`[TPESYSTEM]`
An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file.

`[TPEOS]`
An operating system error has occurred.

See Also

`setscapasswordtool(1)`

tmscd(1)

Name

`tmscd(1)` – Activates and deactivates service contract discovery.

Synopsis

```
tmscd start|stop|status [-e] [-f <file>][id1 [ id2 [ ...]]]
```

Description

The `tmscd` command line utility is used to activate and deactivate service contract discovery.

Parameters and Options

`tmscd` accepts following parameters and options:

`start|stop|status`
Required. Starts, stops, or displays service contract dictionary settings for specific services, or all services if none are specified. A `start` or `stop` request for a service that has already activated or deactivated contract discovery is ignored. Effective service information is displayed when handling the requests.

Note: `start|stop|status` must occur after `-e` and `-f` , if either of those options are specified.

[-e]
Specifies the service scope as a regular expression.

[-f <file>]
The service scope is defined in the given <file>. The file may contain sections to group related definitions together. All entries for a section must be written together line-by-line.

Empty lines or lines starting with '#' are ignored. Lines starting with '*' are section lines. Other lines are "id=content" definitions.

id1 id2 ...
Indicates one or more services. If -e is specified, a regular expression is used to match the service name. If -e is not specified, the service name is matched exactly.

Examples

Example 1 - start discovery for TOUPPER, TOLOWER:

```
tmscd start TOUPPER TOLOWER
```

Example 2 - start discovery for services started with TO and BR:

```
tmscd -e start TO.* BR.*
```

Example 3 - same request as example 1 but via file:

```
tmscd -f svcfile start id1 id2
```

Note: The first found definition is used if section is not provided:

Example 4 - same request as example 2 but via file:

```
tmscd -e -f svcfile start case4.svcs
```

Listing 2 shows content of the file named "svcfile".

Listing 2 svcfile Content

```
# file: svcfile
*case3
id1    = TOUPPER
id2    = TOLOWER

*case4
svcs   = TO.*|BR.*
```

Diagnostics

`tmscd` fails if TMMETADATA is not booted or booted using the `-r` (readonly) option without the `-o` option.

See Also

`TMMETADATA(5)`

Configuring Service Contract Discovery in the *Oracle SALT Administration Guide*

tmwsdlgen

Name

`tmwsdlgen` – WSDL document generator.

Synopsis

```
tmwsdlgen –c wsdf_file [-y] [-o wsdl_file] [-m {pack|raw|mtom}] [-t
{wls|axis}]
```

Description

`tmwsdlgen` generates a WSDL document file from a Tuxedo native Web Service Definition File (WSDF). The generated WSDL document is WSDL 1.1 specification compliant, and represents both the service contracts and policies. `tmwsdlgen` collects Tuxedo service contract information throughout the Tuxedo Service Metadata Repository management (`TMMETADATA`)process. `tmwsdlgen` works as a Tuxedo native client and requires the following:

- the `TUXCONFIG` environment variable must be set correctly
- the relevant Tuxedo application using `TMMETADATA` must be booted prior to executing `tmwsdlgen`.

WARNING: The given WSDF must be a Tuxedo native WSDF. Do not use a `wsdlcvt` converted non-native WSDF file as input.

`tmwsdlgen` accepts the following parameters:

`-c wsdf_file`
Mandatory. Used to specify the SALT WSDF local path.

`tmwsdlgen` accepts the following optional parameters:

-o wsdl_file

Used to specify the output WSDL document file path. If the option is not present, the default file, `tuxedo.wsdl`, is created in the *current directory*. If the specified WSDL document file already exists, then a prompt displays to confirm to overwriting the existing file.

-y

Overwrites the existing WSDL document file without prompting.

-m

Used to specify the WSDL data mapping policy for certain Tuxedo typed buffers. Currently, it applies to the Tuxedo CARRAY buffer type. If `raw` mode is specified, CARRAY is represented to the MIME attachment. If `pack` mode is specified, `xsd:base64Binary` is used to represent CARRAY. The default value is `pack` mode.

Note: `raw` mode cannot be used for .Net clients. The .Net Framework does not support MIME attachments.

If `mtom` is specified, CARRAY is mapped to the MTOM SOAP message.

-t

This option takes effect only when the `-m` option is specified in `raw` mode. It accepts two options, `wls` or `axis`:

- `wls` indicates `tmwsdlgen` generates the WSDL document file in compliance with WebLogic 9.x. The default is `wls`.
- `axis` indicates the WSDL document file format can be recognized by the Apache Axis toolkit.

Deprecation

The following SALT 1.1 `tmwsdlgen` parameters are deprecated in the current release.

-c Config_file

Mandatory. Used to specify the Oracle SALT Configuration File path.

Note: In the current SALT release, the SALT 1.1 configuration file is specified as the `tmwsdlgen` input using the following optional parameters:

-s

Used to specify the encoding style used for Web service SOAP messages. Specifies rpc for RPC/encoded style and doc for Doc/literal encoded style. If this option is not present or the specified value is invalid, Doc is the default style.

-v

Used to specify the SOAP protocol version that the WSDL file supports. Specify 1.1 for SOAP 1.1 protocol and 1.2 for SOAP 1.2 protocol. If this option is not present or the specified value is invalid, SOAP 1.1 is used as the default.

Note: In the current SALT release, the SOAP version and message style attribute are specified in the Oracle SALT WSDF.

Diagnostics

If a syntax error is detected in the given WSDF, an "ERROR" or "FATAL" message indicating that problem is printed to the standard error, and no WSDL file is generated and `tmwsdlgen` exits with exit code 1.

A "WARN" message is printed to the console if:

1. WSDF content may result in a potential run-time risk, or
2. default values are used because they are not specified in the WSDF. "WARN" messages do not interrupt `tmwsdlgen` execution.

Upon successful completion, `tmwsdlgen` exits with exit code 0.

Examples

The following command generates a WSDL document file, `Salt.wsdl`, from the specified SALT WSDF, `tux.wsdf`.

```
tmwsdlgen –c tux.wsdf –o Salt.wsdl
```

The following command generates a default WSDL document file with SOAP w/Attachment capability from the specified SALT WSDF, `app_wsdf.xml`.

```
tmwsdlgen –c app_wsdf.xml –m raw
```

SEE ALSO

GWWS

wsdlcvt

SALT Web Service Definition File Reference

tuxscagen

Name

`tuxscagen` – Generates SCA, SCDL, and server side interface files for Tuxedo services.

Synopsis

```
tuxscagen [-s <target-root-directory>][-d <service-name>]
[-C <TUXEDO_cltname>][-u <TUXEDO_username>][-j <java_package_name>]
[-o <output_SCDL_filename>][-i <output_interface_filename>]
[-m <max-intf-arguments>] [-y][-v][-h] [-F] [-c][-g i|a|s]
[-trepository=<filename> | -tinfile=<metarepos.infile> | -tmetadata]
```

Description

This tool is used to generate interface and SCDL files. The interface files are used for developing the SCA component using ATMI binding, or wrap existing Tuxedo services in an SCA component. The SCDL files are assembly artifacts that help SCA run time to locate the module and services.

Options

`tuxscagen` supports the following options:

`-s target-root-directory`
Specifies the location of the root directory where the generated SCDL and interface files are located. The directory must exist and with write access permission; if it does not exist, the tool issues an error message and fails.

`-d<service-name>`
Specifies the name of Tuxedo service in the Tuxedo Metadata Repository. If this option is not specified, all services in the repository or in the input file are selected.

Abbreviation: there is no abbreviation for this option

`-C <TUXEDO_cltname>`
The Tuxedo client name. Use `cltname` as the client name when joining the Tuxedo application.

`-u <TUXEDO_username>`
The Tuxedo user name. Use `username` as the user name when joining the Tuxedo application. This is required when Tuxedo security level is higher than `APP_PW` and input method is to retrieve Tuxedo Service Metadata from `TUXEDO.TMMETAREPOS` Service.

`-j <java_package_name>`
This option generates JAVA interface files. By default, `tuxscagen` generates C++ header files. If `-g` is not specified but if `-j <java_package_name>` is specified then `-ga` is assumed. However, if `-g` sub-option `i` or `s` is specified, a warning message is displayed.

-o <output_SCDL_filename>

This option specifies the output SCDL filenames for single composite and single `componentType` file. If this option is not specified, then by default, one composite and one `componentType` are generated for each Tuxedo service. However, if this option is specified with the output filename, only one composite and one `componentType` file is generated for all the matching Tuxedo services. If the specified `<output_SCDL_filename>` already exists, an interactive prompt is displayed and requires user input (unless `-y` is specified). If this option is specified, `-F` is automatically implied.

-i <output_interface_filename>

This option specifies the output interface filenames for single abstract class header file and single class implementation header file. If this option is not specified, then by default, it generates one abstract interface class header file and one implementation class header file.

However, if this option is specified with output interface filename then only one abstract class header file and one implementation header file is generated for all matching Tuxedo services. If the specified `<output_interface_filename>` already exists, an interactive prompt is displayed and requires user input (unless `-y` is specified).

If this option is specified, `-F` is automatically implied.

-m <max-intf-arguments>

This option specifies the maximum number of arguments allowed in the interface method. If the number of arguments exceeds the specified threshold then a complex data type is used as the input argument for the interface method. The complex data type used is `commonj::sdo::DataObjectPtr`.

If **`-m`** is not specified, the default threshold is 10.

If `0` specified, it will always generate using `commonj::sdo::DataObjectPtr`.

If `-ga` is not specified, this option is ignored.

-y

This option suppresses `Really overwrite files:<filename> [y, q] ?` so that the script can run without user input. This question appears if either or both `-o` and `-i` are specified. If both these options are not specified, by default existing files are replaced.

-v

This option turns on the verbose mode.

-h

If this option is specified, online help is printed and all other options are ignored.

-F

`Flat File` view. If this option is specified, then all the generated files are put in the target root directory. The default is `Tree File` view.

-c

Generates client-side SCDL. By default `tuxscagen` generates server-side SCDL, specifying this option changes it to generate client-side SCDL.

-g a|i|s

This option is used to specify the files to generate. The sub-options can be combined. The `a` sub-option is used to generate abstract base class header files. The sub-option `i` is to generate implementation class header files. Sub-option `s` is used to generate SCDL files. To generate both header files, specify `-gai`. To generate all files, specify `-gais`.

If not specified, `-gais` is assumed.

[-trepository=<filename> | -tinfile=<metarepos.infile> | -tmetadata]

This option specifies the processing type.

If `-trepository=<filename>` is specified, `tuxscagen` retrieves service parameter information from the Service Metadata repository file `<filename>`.If `-tinfile=<metarepos.infile>` is specified, then `tuxscagen` retrieves service parameter information from `<metarepos.infile>`, where the `<metarepos.infile>` syntax is suitable for input to `tmloadrepos`. If `-tmetadata` is specified, `tuxscagen` retrieves service parameter information from the Tuxedo TMMETADATA server.

At most, one `-t` option can be specified; the default is `-tmetadata`.

Portability

This tool is available on ny platform on which the Oracle SALT environment is supported.

Example

The following command is used to generate SCDL, interface, and implementation header files from a Tuxedo Metadata Repository file named `myrepository` in the current working directory. The number of interface method input arguments is limited to 8. If the limit is exceeded, the XSD schema file is still generated.

```
tuxscagen -s /home/tux/sca -Dname=TRANSFER -gais -m 8
-trepository=myrepository
```

See Also

`tmloadrepos(1), tmunloadrepos(1)`

Managing The Tuxedo Service Metadata Repository in *Setting up a BEA Tuxedo Application.*

wsadmin

Name

wsadmin – Oracle SALT administration command interpreter.

Synopsis

```
wsadmin [-v]
```

Description

wsadmin uses specific commands to monitor and administrate active GWWS processes in the specified Tuxedo domain. The TUXCONFIG environment variable is used to determine the location where the Tuxedo configuration file is loaded. wsadmin is used in the same manner as tmadmin(1) or dmadmin(1).

wsadmin accepts below optional parameter:

-v
: Causes wsadmin to display the Oracle SALT version number, SALT Patch Level and license information. wsadmin exits after print out.

wsadmin Commands

Commands may be entered using either their full name or their abbreviation (as given in parentheses), followed by any appropriate arguments. Arguments appearing in brackets, [ ], are optional; arguments in braces, {}, indicate a selection from mutually exclusive options.

Note: Command line options that are not in brackets do not need to appear in the command line if the corresponding default has been set via the default command.

wsadmin supports the following commands:

configstats(cstat) -i gwws_instance_id
: Displays the current configuration status for the specified GWWS process. The -i parameter must be specified.

default(d) [-i gwws_instance_id]
: Sets the corresponding argument to the default GWWS Instance ID. The defaults can be changed by specifying * as an argument. If the default command is entered without arguments, the current defaults are printed.

echo(e) [{off | on}]
: Echoes input command lines when set to on. If no option is given, the current setting is toggled, and the new setting is printed. The initial setting is off.

help (h) [command]
Prints help messages. If command is specified, the abbreviation, arguments, and description for that command are printed.

Omitting all arguments causes the syntax of all commands to be displayed.

gwstats(gws) -i gwws_instance_id [-s serviceName]
Displays global level run time statistics information for the specified GWWS processes including fail, success, pending number for both inbound and outbound call, average processing time, active thread number, etc. If -s `serviceName` specified, the server-level information is displayed.

-i is mandatory. **-s** is optional.

paginate(page) [{off | on}]
Paginates output. If no option is given, the current setting is toggled, and the new setting is printed. The initial setting is on, unless either standard input or standard output is a non-tty device. Pagination may be turned on only when both standard input and standard output are tty devices.

The default paging command is indigenous to the native operating system environment. In a UNIX operating system environment, for example, the default paging command is `pg`. The shell environment variable PAGER may be used to override the default command used for paging output

quit (q)
Terminates the session.

verbose (v) [{off | on}]
Produces output in verbose mode. If no option is given, the current setting is toggled, and the new setting is printed. The initial setting is off.

! shellcommand
Escapes to the shell and executes shell command.

! !
Repeats previous shell command.

 # [text]
Specifies comments. Lines beginning with # are ignored.

<CR>
Repeats the last command.

Examples

1. The following command inspects run time statistics for both inbound and outbound service on GW2:

```
wsadmin
> gws -i GW2
GWWS Instance : GW2
Inboud Statistics :
---------------------------------
Request Response Succ :   3359
Request Response Fail :      0
              Oneway Succ :      0
              Oneway Fail :      0

              Total Succ :   3359
              Total Fail :      0

Avg. Processing Time : 192.746 (ms)
---------------------------------

Outboud Statistics :
---------------------------------
Request Response Succ :   4129
Request Response Fail :      0
              Oneway Succ :      0
              Oneway Fail :      0

              Total Succ :   4129
              Total Fail :      0

Avg. Processing Time : 546.497 (ms)
---------------------------------
Total request Pending :     36
Outbound request Pending :      0
Active Thread Number :    141
```

The following command inspects run time statistics for the ToUpperWS service on GW1 and gets output in verbose mode.

```
wsadmin
> > verbose
Verbose now on.
> gws -i GW1 -s ToUpperWS
GWWS Instance : GW1
Service : ToUpperWS
Outboud Statistics :
---------------------------------
              Oneway Succ :      0
              Oneway Fail :      0
--------------------------------
Avg. Processing Time :  0.000 (ms)
```

See Also

GWWS

SALT Administration Guide

wsdlcvt

Name

`wsdlcvt` – WSDL document converter.

Synopsis

```
wsdlcvt -i WSDL_URL -o output_basename [-m] [-v] [-y] [-w]
```

Description

`wsdlcvt` is used to convert an existing WSDL 1.1 document to a Metadata Input File, FML32 mapping File and Oracle SALT Web Service Definition File (WSDF). It is a wrapper script for `wsdl2mif.xsl`, `wsdl2fml32*.xsl` and `wsdl2wsdf.xsl` for Xalan. Apache Xalan 2.7 libraries are bundled with Oracle SALT product.

JRE 1.5 or higher is required to run `wsdlcvt`.

Parameters

`wsdlcvt` accepts the following parameters:

-i

Specifies the URL of the input WSDL document. The URL can be a local file path or a downloadable HTTP URL link.

-o

Specifies the output files basename. The suffixes shown in Table 4 are appended after the basename:

Table 4 wsdlcvt-Created File Suffixes

Suffix	Output File
`.mif`	Tuxedo Service Metadata Input File
`.fml32`	FML32 Field Table Definition File
`.wsdf`	SALT Web Service Definition File
`.xsd`	The WSDL Document embedded XML Schema File

`wsdlcvt` accepts the following optional parameters:

-y

Specifies that all the output destination files are overwritten without prompting if they exist. If this parameter is not specified, a prompt message is output.

-m

Specifies that the "`xsd:string`" data type is mapped to an FML32 typed buffer Tuxedo `FLD_MBSTRING` data type. If this parameter is not specified, Tuxedo `FLD_STRING` data type is mapped by default.

-v

Specifies that `wsdlcvt` works in verbose mode. In particular, it shows context information in the message and output context as FML32 field comments.

-w

If the given WSDL document is published using Microsoft .NET WCF, specifies this parameter to ensure `wsdlcvt` can handle it correctly.

Environment Variables

The `TUXDIR` and `LANG` environment variables must be set correctly.

The `PATH` environment variable must be set appropriately to execute "java".

Diagnostics

Error, warning or information messages are output to standard output.

Examples

The following command converts the local WSDL file, `sample.wsdl`.

```
wsdlcvt -i sample.wsdl -o sample
```

The following command converts a WSDL document from a HTTP URL link. The "`xsd:string`" data type is mapped to the Tuxedo `FLD_MBSTRING` field type.

```
wsdlcvt -i http://api.google.com/GoogleSearch.wsdl -o GSearch -m
```

See Also

Creating The Tuxedo Service Metadata Repository

field_tables(5)

SALT Web Service Definition File Reference

wsloadcf

Name

`wsloadcf` – Reads SALT Deployment file and other referenced artifacts. Loads a binary `SALTCONFIG` file.

Synopsis

```
Usage 1: wsloadcf [-n][-y][-D loglevel] saltdeploy_file
Usage 2: wsloadcf [-n][-y][-D loglevel] -1 [-s rpc|doc]
[-v 1.1|1.2] salt_1.1_config
```

Description

`wsloadcf` reads a SALT deployment file and other referenced files (WSDF files, WS-Policy files), checks the syntax, and optionally loads a binary `SALTCONFIG` file. The `SALTCONFIG` environment variable points to the `SALTCONFIG` file where the information should be stored. The generated `SALTCONFIG` file is necessary to boot `GWWS` servers.

`wsloadcf` accepts the following optional parameters:

`-n`

Do validation only *without* generating the `SALTCONFIG` file.

-y

After checking the syntax, `tmloadcf` checks whether: (a) the file referenced by `SALTCONFIG` exists; (b) it is a valid Oracle Tuxedo system file system; and (c) it contains `SALTCONFIG` tables. If these conditions are not true, `wsloadcf` prompts you to indicate whether you want the command to create and initialize `SALTCONFIG`.

```
Initialize SALTCONFIG file: path [y, q]?
```

Prompting is suppressed if the -y option is specified on the command line.

-D

Used to specify the configuration parsing log level.

For SALT 1.1 backward compatibility, `wsloadcf` can also read a SALT 1.1 configuration file. Besides generating the `SALTCONFIG` binary file, `wsloadcf` also generates one SALT Web Service Definition File (WSDF) and one SALT Deployment file according to the given SALT 1.1 configuration file.

-1

Turns on the SALT 1.1 compatible mode. To pass the SALT 1.1 configuration file to `wsloadcf`, you must specify this flag first.

-v

Only takes effect when a SALT 1.1 configuration file is used. This option is used to specify which SOAP version is applied to the generated WSDF file.

-s

Only takes effect when a SALT 1.1 configuration file is used. This option is used to specify which SOAP message style is applied to the generated WSDF file.

Environment Variables

The `SALTCONFIG` environment variable must be set before executing `wsloadcf`.

Diagnostics

If a syntax error is detected in the given configuration files, an “ERROR” or “FATAL” message indicating that problem is printed to the console, and no information is updated in the `SALTCONFIG` file. `wsloadcf` exits with exit code 1.

A “WARN” message is printed to the console if: (1) configuration files may result in a potential run-time risk or (2) default values are used because they are not specified in the configuration files. “WARN” messages do not interrupt `wsloadcf` execution.

Upon successful completion, `wsloadcf` exits with exit code 0. If the `SALTCONFIG` file is updated, a userlog message is generated.

See Also

SALT Web Service Definition File Reference

SALT Deployment File Reference

APPENDIX A

Oracle SALT Web Service Definition File Reference

The following sections provide SALT Web Service Definition File (WSDF) reference information:

Overview

The Oracle SALT Web Service Definition File (WSDF) is an XML-based file used to define Oracle SALT Web service components (for example, Web Service Bindings, Web Service Operations, Web Service Policies, and so on). WSDF is a SALT specific representation of the Web Service Definition Language data model. There are two WSDF types:

- Native WSDF (Tuxedo generated)

  A native WSDF is composed manually. You must define a set of Tuxedo services and how they are exposed as Web services in a native WSDF. The native WSDF is similar to the SALT 1.1 configuration file.

  Note: A native WSDF is the input file used by the SALT WSDL generator (tmwsdlgen).

- Non-native `WSDF` (Externally generated)

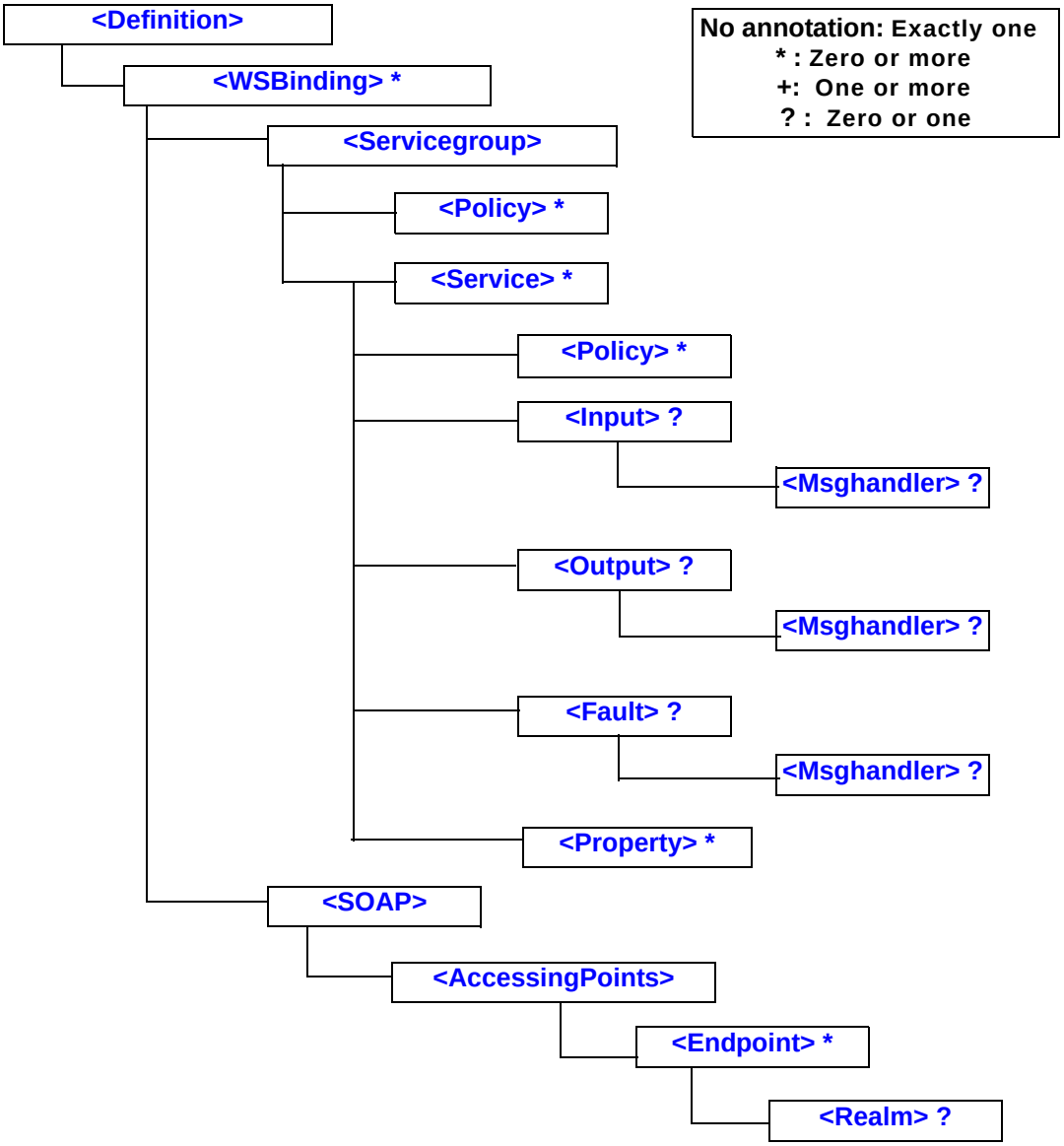
  A non-native `WSDF` is generated from an external WSDL file via the SALT WSDL converter (`wsdlcvt`). In most cases, you do not need to change the generated WSDF except for configuring advanced features.

For more information, see `tmwsdlgen` and `wsdlcvt` in the *Oracle SALT Command Referenc*e.

Oracle SALT WSDF Format

Figure A-1 shows a graphical representation of the WSDF format.

Figure A-1 SALT Web Service Definition File Format



XML Schema

An XML Schema is associated with the WSDF. The XML Schema file that describes the WSDF format is located in the following directory: $TUXDIR/udataobj/salt/wsdf.xsd.

Oracle SALT WSDF Examples

Listing A-1 and Listing A-2 show native and non-native WSDF examples.

Listing A-1 Native WSDF (Composed Manually)

```
<Definition name="bankapp"
  xmlns=http://www.bea.com/Tuxedo/WSDF/2007 >
  <WSBinding id="bankapp_binding" >
    <Servicegroup id="bankapp">
      <Policy location="/home/user/rm.xml" />
      <Service name="inquiry" />
      <Service name="deposit" />
    </Servicegroup>
    <SOAP>
      <AccessingPoints>
        <Endpoint id="HTTP1" address="http://myhost:7001" />
        <Endpoint id="HTTPS1" address="https://myhost:7002/bankapp" />
      </AccessingPoints>
    </SOAP>
  </ WSBinding >
</Definition>
```

Listing A-2 Non-Native WSDF (Generated from an External WSDL Document)

```
<Definition name="myWebservice"
  wsdlNamespace="http://www.example.org/myWebservice"
  xmlns=http://www.bea.com/Tuxedo/WSDF/2007 >
  <WSBinding id="A_binding">
    <Servicegroup id="portType">
      <Service name="operation_1" soapAction="op1" />
```

```
      <Service name="operation_2" soapAction="op2" />
    </Servicegroup>
    <SOAP version="1.1" style="rpc" use="encoded">
      <AccessingPoints>
        <Endpoint id="example_http_port"
                  address="http://www.example.org/abc" />
        <Endpoint id="example_https_port"
                  address="https://www.example.org/abcssl" />
      </AccessingPoints>
    </SOAP>
  </WSBinding>
  <WSBinding id="B_binding">
    <Servicegroup id="portType">
      <Service name="operation_3" soapAction="op3" />
      <Service name="operation_4" soapAction="op4" />
    </Servicegroup>
    <SOAP version="1.2">
      <AccessingPoints>
        <Endpoint id="another_http_port"
                  address="http://www.example.org/def" />
      </AccessingPoints>
    </SOAP>
  </WSBinding>
</Definition>
```

Oracle SALT WSDF Element Descriptions

WSDF format elements and their attributes are listed and described in the following section.

<Definition>

Table A-1 lists the `WSDF` file root elements and attributes.

Table A-1 <Definition> Attributes

Attribute	Description	Required
`name`	The `WSDF` name. This attribute value may contain a maximum of 30 characters (excluding the terminating NULL character). Native `WSDF`: you must manually provide a distinct application name. Non-native `WSDF`: this value is the same as the WSDL converter (wsdlcvt) command line input parameter "`output_basename`.	Yes
`wsdlNamespace`	The corresponding WSDL document target namespace for the WSDF. Native WSDF: you can optionally specify a distinct URI string so that the generated WSDL can use this as the target namespace. If not specified, the default WSDL target namespace is as follows: "`urn:<wsdf_name>.wsdl`". For example, if the WSDF name is "*`simpapp`*", then the default WSDL target namespace is "*`urn:simpapp.wsdl`*". Non-native WSDF: the value is the WSDL target namespace of the external WSDL document.	No

<WSBinding>

Defines concrete protocol binding information. Zero or more WSBinding objects can be specified in one WSDF file.

Native WSDF: you can set SOAP version, encoding style, several endpoints for Web Service Client connection through sub element <SOAP> and a set of Tuxedo services to be exposed for invocation through sub element <Servicegroup>.

Non-native WSDF: each SOAP binding object (i.e., `wsdl:binding` object with `soap:binding` extension) in the external WSDL document is translated into one WSBinding object. Table A-2 lists the <WSBinding> attributes.

Table A-2 <WSBinding> Attributes

Attribute	Description	Required
`id`	Identifies the WSBinding object. The value must be unique within the WSDF. This attribute value may contain a maximum of 78 characters (excluding the terminating NULL character). Native WSDF: the value is specified by customers and is used as the `wsdl:binding` name in the generated WSDL document. Non-native WSDF: the value is the `wsdl:binding` name defined in the external WSDL document.	Yes

<Servicegroup>

Defines a Servicegroup object for one WSBinding object. Each WSBinding object must have exactly one Servicegroup. The Servicegroup object is used to encapsulate a set of Tuxedo services. Table A-3 lists the <Servicegroup> attributes.

Table A-3 <Servicegroup> Attributes

Attribute	Description	Required
`id`	Specifies the service group id. This attribute value may contain a maximum of 78 characters (excluding the terminating NULL character). Native WSDF: the value is specified by customers and is used as the `wsdl:portType` name in the generated WSDL document. Non-native WSDF: the value is the `wsdl:portType` name defined in the external WSDL document.	Yes

<Service>

Specifies a service for the WSBinding object.

Native WSDF: each service is a Tuxedo service.

Non-native WSDF: each service represents a converted Tuxedo service from a `wsdl:operation` object defined in the external WSDL document.Table A-4 lists the <Service> attributes.

Table A-4 <Service> Attributes

Attribute	Description	Required
`name`	Specifies the service name. This attribute value may contain a maximum of 256 characters (excluding the terminating NULL character). Native WSDF: the service name value is used as the `wsdl:operation` name in the generated WSDL document. Non-native WSDF: the service name is equal to the `wsdl:operation` name defined in the external WSDL document.	Yes
`tuxedoRef`	An optional attribute used to reference the service definition in the Tuxedo Service Metadata Repository. If not specified, attribute "`name`" value is used as the reference value.	No
`soapAction`	Specifies the service soapAction attribute. This is a *non-native* WSDF attribute. It is used to save the soapAction setting for each `wsdl:operation` defined in the external WSDL document. **Note:** Do not specify this attribute for a native WSDF.	No
`namespace`	Specifies service namespace attribute. This is a *non-native* WSDF attribute. It is used to save the namespace setting for each `wsdl:operation` defined in the external WSDL document. **Note:** Do not specify this attribute for a native WSDF.	No

<Input>

Specifies Input message attributes for a particular service. This element is optional.Table A-5 lists the <Input> attributes.

Table A-5 <Input> Attributes

Attribute	Description	Required
`name`	Specifies the service input message name attribute. This is a *non-native* WSDF attribute. It is used is used to save the name for the input `wsdl:message` defined in the external WSDL document. **Note:** Do not specify this attribute for a native WSDF.	No
`wsaAction`	Specifies the service input message wsaAction attribute. This is a *non-native* WSDF attribute. It is used is used to save the wsaAction attribute of the input `wsdl:message` defined in the external WSDL document. **Note:** Do not specify this attribute for a native WSDF.	No

<Output>

Specifies Output message attributes for a particular service. This element is optional.Table A-6 lists the <Output> attributes.

Table A-6 <Output> Attributes

Attribute	Description	Required
`name`	Specifies the service output message name attribute. This is a *non-native* WSDF attribute. It is used to save the name for the output `wsdl:message` defined in the external WSDL document. **Note:** Do not specify this attribute for a native WSDF.	No
`wsaAction`	Specifies the service output message name attribute. This is a *non-native* WSDF attribute. It is used to save the wsaAction attribute of the output `wsdl:message` defined in the external WSDL document. **Note:** Do not specify this attribute for a native WSDF.	No

<Fault>

Specifies Fault message attributes for a particular service. This element is optional.Table A-7 lists the <Fault> attributes.

Table A-7 <Fault> Attributes

Attribute	Description	Required
`name`	Specifies the service fault message name attribute. This is a *non-native* WSDF attribute. It is used to save the name for the fault `wsdl:message` defined in the external WSDL document. **Note:** Do not specify this attribute for a native WSDF.	No
`wsaAction`	Specifies the service fault message wsaAction attribute. This is a *non-native* WSDF attribute. It is used to save the wsaAction attribute of the fault `wsdl:message` defined in the external WSDL document. **Note:** Do not specify this attribute for a native WSDF.	No

<Msghandler>

Specifies a customized message conversion handler. Optional for <Input>, <Output> and/or <Fault> elements of any service. The value of this element is the handler name, which may contain a maximum of 30 characters (excluding the terminating NULL character).

The GWWS server looks for the message conversion handler from all known message conversion plug-in shared libraries using the handler name.The message conversion handler allows you to develop customized Tuxedo buffer and SOAP message payload transformation functions to replace the default GWWS message conversions.

For more information, see "Programming Message Conversion Plug-ins"in the *Oracle SALT Programming Web Services*.

<Policy>

References one Web Service Policy file applied to one of the following two levels:

- <Servicegroup> level
- <Service> level

At most, 10 Web Service policies can be referenced for each object.Table A-8 lists the <Policy> attributes.

Table A-8 <Policy> Attributes

Attribute	Description	Required
`location`	Specifies the local file path for the referenced WS-Policy file. This attribute value may contain a maximum of 256 characters (excluding the terminating NULL character). Specifically, Oracle SALT pre-defines WS-Policy template files for typical WS-* scenarios. These files can be found under the `$TUXDIR/udataobj/salt/policy` directory. You can reference these template files using the string format "`salt:<template_file_name>`". For example, if you want to reference SALT WS-SecurityPolicy 1.0 template file "`wssp1.0-signbody.xml`", you should define the following XML snippet in the WSDF file: <Policy location="salt:wssp1.0-signbody.xml" />	Yes
use	Specifies if the WS-Policy file is applied to the input message, output message, fault message, or the combination of the three. If multiple messages are set, use a space as the delimiter. For example, if you want to configure a WS-Policy file "mypolicy.xml" to be applied to "input" and "output" messages, you should define the following XML snippet in the WSDF file: <Policy location="mypolicy.xml" use="input output"/> Oracle SALT limits the applicable messages for each supported WS-Policy assertion. For more information, see the following sections: • "Configuring Advanced Web Service Messaging Features" in the *Oracle SALT Administration Guide* • "Configuring Message-Level Web Service Security" in the *Oracle SALT Administration Guide* • Oracle SALT WS-ReliableMessaging Policy Assertion Reference • Oracle SALT WS-SecurityPolicy Assertion 1.2 Reference • Oracle SALT WS-SecurityPolicy Assertion 1.0 Reference	No

<Property>

Specifies SALT specific properties for each service object.Table A-9 lists the <Property> attributes.

Table A-9 <Property> Attributes

Attribute	Description	Required
`name`	Specifies the property name. Table A-10 lists all the GWWS server properties.	Yes
`value`	Specifies the property value.	Yes

Table A-10 lists all properties that can be specified for each service object.

Table A-10 <Property> Name List

Property	Description	Values
`async_timeout`	Outbound service: Specifies a time setting to wait for SOAP response. Inbound service: No behavior impact.	`(0-32767] (sec)` Default: `60 secs`.
`disableWSAddressing`	Outbound service: Disables explicit Web Service Addressing requests with this property. Inbound service: No behavior impact.	`{True|False}` Default: `False`

<SOAP>

Specifies SOAP protocol information for the WSBinding object. SOAP version, message style accessing endpoints are specified in this element.Table A-11 lists the <SOAP> attributes.

Table A-11 <SOAP> Attributes

Attribute	Description	Required
`version`	Specifies SOAP version for this WSBinding object. The valid values are “`1.1`” and “`1.2`”. If not specified, "`1.1`" is used.	No

Table A-11 <SOAP> Attributes

Attribute	Description	Required
`style`	Specifies SOAP message style for this WSBinding object. The valid values are "`rpc`" and "`document`". If not specified, "`document`" is used.	No
`use`	Specifies SOAP message encoding style for this WSBinding object. The valid values are "`encoded`" and "`literal`". If not specified explicitly, this value is automatically selected according to "`style`" value. If "`style`" is "`rpc`", then "`encoded`" is used; if "`style`" is "`document`", then "`literal`" is used.	No

Note: In the current SALT release, only "`rpc/encoded`" and "`document/literal`" are supported.

<AccessingPoints>

Specifies the endpoint list for the WSBinding object. Each sub element <Endpoint> represents one particular endpoint.

There are no attributes for this element.

<Endpoint>

Specifies each accessing endpoint for the WSBinding object.Table A-12 lists the <Endpoint> attributes.

Table A-12 <Endpoint> Attributes

Attribute	Description	Required
`id`	Specifies a unique endpoint id value within the WSBinding object. This attribute value may contain a maximum of 78 characters (excluding the terminating NULL character).	Yes
`address`	Specifies the endpoint address. The address value must use the following format: "`http(s)://<host>:<port>/<context_path>`" **Note:** Two endpoints cannot be specified with exact the same address URL value.	Yes

<Realm>

Specifies the HTTP Realm attribute of an HTTP and/or HTTP/S endpoint. If this element is configured for one endpoint, the GWWS tries to incorporate HTTP basic authentication information in the request messages when issuing outbound calls through this endpoint.

For more information, see "Configuring Transport Level Security" in the *Oracle SALT Administration Guide*.

Note: This element only works for non-native (external) WSDF files.

APPENDIX B

Oracle SALT Deployment File Reference

The following sections provide SALT Deployment File reference information

Overview

The Oracle SALT Deployment File (SALTDEPLOY) is an XML-based file used to define Oracle SALT `GWWS` server deployment information on a per Tuxedo machine basis. SALTDEPLOY does the following:

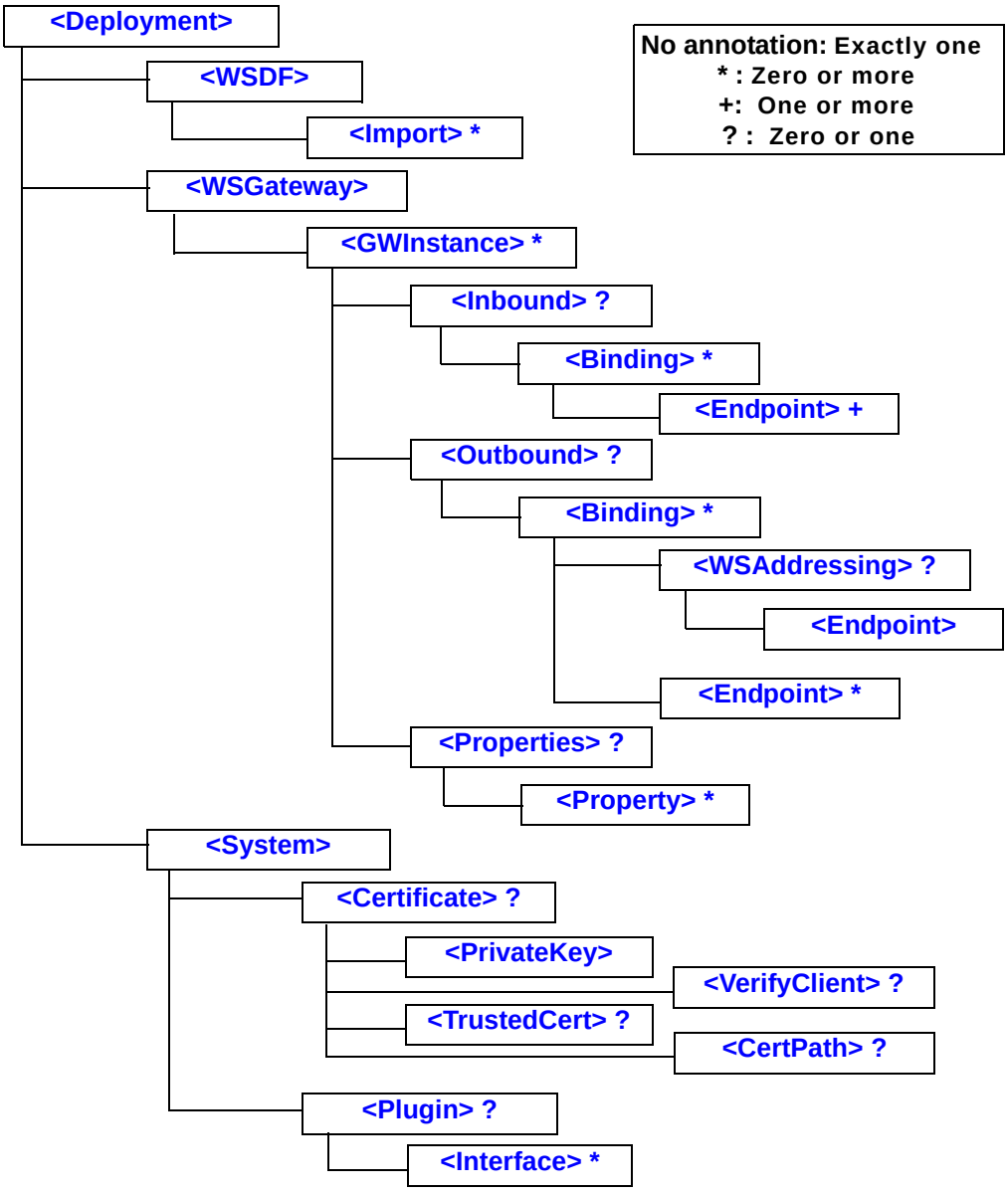
- lists all necessary Web Service Definition Files (WSDF)
- specifies how many `GWWS` servers are deployed on a Tuxedo machine
- associates inbound and outbound Web Service access endpoints for each `GWWS` server.

SALTDEPLOY also provides a system section to configure global resources (for example certificates, plug-in load libraries, and so on).

Oracle SALT SALTDEPLOY Format

Figure B-1 shows a graphical representation of the Oracle SALT SALTDEPLOY format.

Figure B-1 SALT Deployment File Format



XML Schema

An XML Schema is associated with an Oracle SALT Deployment File. The XML Schema file that describes the Oracle SALT Deployment File format is located in the following directory: `$TUXDIR/udataobj/salt/saltdep.xsd`.

Oracle SALT SALTDEPLOY Example

Listing B-1 shows a sample SALT Deployment File.

Listing B-1 SALT Deployment File Example

```
<Deployment xmlns=”http://www.bea.com/Tuxedo/SALTDEPLOY/2007”>
  <WSDF>
    <Import location="/home/myapp/bankapp.wsdf" />
    <Import location="/home/myapp/amazon.wsdf" />
  </WSDF>
  <WSGateway>
    <GWInstance id="GW1">
      <Inbound>
        <Binding ref="bankapp:bankapp_binding">
          <Endpoint use="http1"/>
          <Endpoint use="https1" />
        </Binding>
      </Inbound>
      <Outbound>
        <Binding ref="amazon:default_binding"/>
      </Outbound>
    </GWInstance>
  </WSGateway>
  <System>
    <Certificate>
      <PrivateKey>/home/user/cert.pem</PrivateKey>
    </Certificate>
    <Plugin>
      <Interface library=”/home/user/mydatahandler.so” />
    </Plugin>
```

```
  </System>
</Deployment>
```

Oracle SALT SALTDEPLOY Element Description

SALTDEPLOYF format elements and their attributes are listed and described in the following section.

<Deployment>

The SALTDEPLOY file root element.

There is no attribute for this element.

Three sections must be defined within the <Deployment> element:

- <WSDF> elements
- <WSGateway> element
- <System> element.

There can be only one <Deployment> element defined in a SALTDEPLOY file.

<WSDF>

Top element that encapsulates all imported WSDF files.

There is no attribute for this element.

<Import>

Specifies the WSDF to be imported in the SALTDEPLOY file. Multiple WSDF can be imported at the same time. Each WSDF file can only be imported once. Multiple WSDF with the same WSDF name cannot be imported in the same SALTDEPLOY file. Table B-1 lists the <Import> attributes.

Table B-1 <Import> Attributes

Attribute	Description	Required
location	Specifies the WSDF local file path.	Yes

<WSGateway>

Top element that encapsulates all GWWS instance definitions.

There are no attributes for this element.

<GWInstance>

Specifies a single GWWS instance. Table B-2 lists the <GWinstance> attributes.

Table B-2 <GWInstance> Attributes

Attribute	Description	Required
id	Specifies the GWWS identifier. This attribute value may contain a maximum of 12 characters (excluding the terminating NULL character). The identifier value must be unique within the SALTDEPLOY file.	Yes

<Inbound>

Specifies inbound WSBinding objects for the GWWS server. Each inbound WSBinding object is specified using the <Binding> sub element.

There is no attribute for this element.

<Outbound>

Specifies outbound WSBinding objects for the GWWS server. Each outbound WSBinding object is specified using the <Binding> sub element.

There are no attributes for this element.

<Binding>

Specifies a concrete WSBinding object as either an inbound or outbound binding, depending on the parent element. Table B-3 lists the <Binding> attributes.

Table B-3 <Binding> Attributes

Attribute	Description	Required
ref	Specifies a concrete WSBinding object using the following Qualified Name format: "<WSDF_name>:<WSBinding_id>"	Yes

Note: Please note the following maximum WSBinding object limitations for each GWWS server:

- Each GWWS server may reference at most 64 inbound WSBinding objects.
- Each GWWS server may reference at most 128 outbound WSBinding objects.

For TCP/IP addresses, one of the following formats is used as shown in Table B-4.

Table B-4 Ipv4 and IPv6 Address Formats

IPv4	IPv6
//IP:port	//[IPv6 address]:port
//hostname:port_number	//hostname:port_number
//#.#.#.#:port_number	Hex format is not supported

For more information, see TMUSEIPV6 in the TUXENV(5) environment variable listing found in the *Tuxedo 10g R3 Reference Guide, Section 5 - File Formats, Data Descriptions, MIBs, and System Processes Reference*.

<Endpoint>

Specifies a single WSBinding objects endpoint reference.

If the referenced endpoint is specified as an inbound endpoint, the GWWS server creates the corresponding HTTP and/or HTTPS listen endpoint. At least one inbound endpoint must be specified for one inbound WSBinding object.

If the referenced endpoint is specified as an outbound endpoint, the GWWS server creates HTTP and/or HTTPS connections per SOAP requests for the outbound WSBinding object.

If an outbound endpoint is not specified for the outbound WSBinding object, the first 10 endpoints (at most) are auto-selected.

The referenced endpoint must already be defined in the WSDF.Table B-5 lists the <Endpoint> attributes.

Table B-5 <Endpoint> Attributes

Attribute	Description	Required
use	The referenced endpoint id defined in the WSDF.	Yes

Note: Please note the following maximum endpoints limitations for each GWWS server:

- Each GWWS server may create at most 128 inbound endpoints in all inbound WSBinding objects to accept SOAP requests.
- Each GWWS server may create connectivity with at most 256 outbound endpoints in all outbound WSBinding objects.

<WSAddressing>

Specifies if Web Service Addressing is enabled for the outbound WSBinding object.

If this element is present, by default all SOAP messages are sent out with a Web Service Addressing message header. The sub element <Endpoint> must be specified for the listen endpoint address if this element is present.

There is no attribute for this element.

<Endpoint>

Specifies the WS-Addressing listen endpoint address for the referenced outbound WSBinding object. Table B-6 lists the <Endpoint> attributes.

Table B-6 <Endpoint> Attributes

Attribute	Description	Required
`address`	Specifies the WS-Addressing listen endpoint address. The address value must be in the following format: "`http(s)://<host>:<port>/<context_path>`" The GWWS server creates listen endpoints and usage for receiving WS-Addressing SOAP response messages.	Yes

<Properties>

Top element that encapsulates all `GWWS` server property settings using the <Property> sub element.

There are no attributes for this element.

<Property>

Specifies a single `GWWS` property. Table B-7 lists the <Property> attributes. Table B-8 shows the <Property> listings.

Table B-7 <Property> Attributes

Attribute	Description	Required
`name`	Specifies the property name. Table B-8 lists all the GWWS server properties.	Yes
`value`	Specifies the property value.	Yes

Table B-8 GWWS <Property> Listings

Property	Description	Values
max_content_length	Enables the GWWS server to deny the HTTP requests when the content length is larger than the property setting. If not specified, the GWWS server does not check for it. The string value can be one of the following three formats: 1. Integer number in bytes. No suffix means the unit is bytes. 2. Float number in kilobytes. The suffix must be 'K'. For instance, 10.4K, 40K, etc. 3. Float number in megabytes. The suffix must be 'M'. For instance, 100M, 20.6M, etc.	The equivalent byte size value must be in [1 byte, 1G byte] range.
thread_pool_size	Specifies the maximum thread pool size for the GWWS server. **Note:** This value defines the maximum possible threads that may be spawned in the GWWS server. When the GWWS server is running, the actual spawned threads may be less than this value.	The valid value is in [1, 1024]. Default value: 16
timeout	Specifies the network time-out value, in seconds.	The valid value is in [1, 65535]. Default value: 300
max_backlog	Specifies the backlog listen socket value. It controls the maximum queue length of pending connections by operating system. **Note:** Generally no tuning is needed for this value.	The valid value is [1-255]. Default value: 16

Table B-8 GWWS <Property> Listings

Property	Description	Values
`enableMultiEncoding`	Toggles on/off multiple encoding message support for the GWWS server. If multiple encoding support property is turned off, only UTF-8 HTTP / SOAP messages can be accepted by the GWWS server.	The valid values are "`true`", "`false`". Default value: `false`
`enableSOAPValidation`	Toggles on/off XML Schema validation for inbound SOAP request messages if the corresponding Tuxedo input buffer is associated with a customized XML Schema.	The valid values are "`true`", "`false`". Default value: `false`

<System>

Specifies global settings, including certificate information, plug-in interfaces.

<Certificate>

Specifies global certificate information using sub elements <PrivateKey>, <VerifyClient>, <TrustedCert> and <CertPath>.

There are no attributes for this element.

<PrivateKey>

Specifies the PEM format private key file. The key file path is specified as the text value for this element. The server certificate is also stored in this private key file. The value of this element may contain a maximum of 256 characters (excluding the terminating NULL character).

This element is mandatory if the parent <Certificate> element is configured.

<VerifyClient>

Specifies if Web service clients are required to send a certificate via HTTP over SSL connections. The valid element values are "`true`" and "`false`".

This element is optional. If not specified, the default value is "`false`".

<TrustedCert>

Specifies the file name of the trusted PEM format certificate files. The value of this element may contain a maximum of 256 characters (excluding the terminating NULL character).

This element is optional.

<CertPath>

Specifies the local directory where the trusted certificates are located. The value of this element may contain a maximum of 256 characters (excluding the terminating NULL character).

This element is optional.

Note: If <VerifyClient> is set to "`true`", or if WS-Addressing is used with SSL, trusted certificates must be stored in the directory setting with this element.

<Plugin>

Specifies the global plug-in load library information. Each <Interface> sub element specifies one plug-in library to be loaded.

There is no attribute for this element.

<Interface>

Specifies one particular plug-in interface or a plug-in library for all plug-in interfaces inside the library. Table B-9 lists the <Intereface> attributes.

Table B-9 <Interface> Attributes

Attribute	Description	Required
`library`	Mandatory. Specifies a local shared library file path. This attribute value may contain a maximum of 256 characters (excluding the terminating NULL character).	Yes
`params`	Optional. Specifies a particular string value that is passed to the library when initialized by the `GWWS` server at boot time. This attribute value may contain a maximum of 256 characters (excluding the terminating NULL character).	No

Note: For more information about how to develop a SALT plug-in interface, see "Using Oracle SALT Plug-ins" in the *Oracle SALT Programming Web Services*.

APPENDIX C

Oracle SALT WS-ReliableMessaging Policy Assertion Reference

The following sections provide SALT WS-ReliableMessaging (WS-RM) Policy reference information:

Overview

Oracle SALT provides support for WS-ReliableMessaging (WS-ReliableMessaging 1.0, Feb., 2005 specification), which allows two Web Service applications running on different GWWS instances to communicate reliably in the event of software component, system, or networks failure.

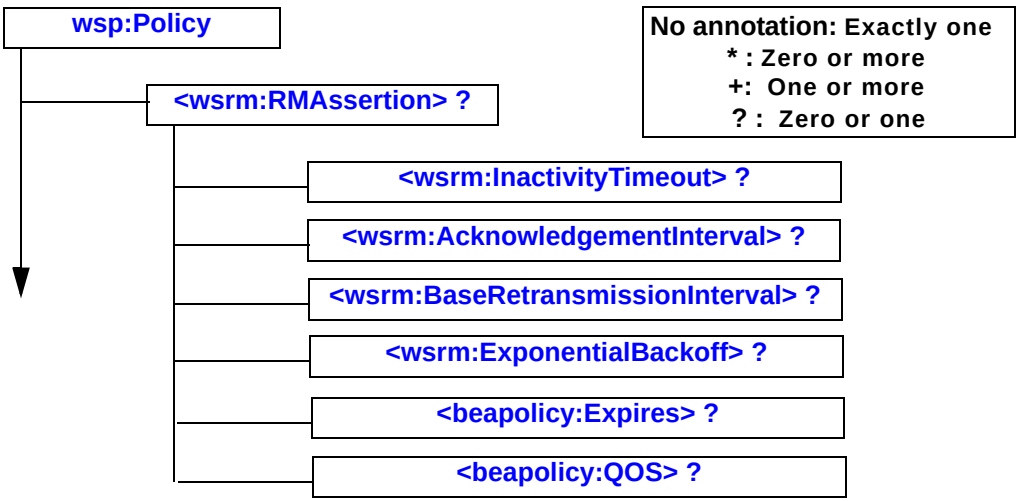
A WS-Policy file containing WS-ReliableMessaging Policy Assertion is used to configure the reliable messaging capabilities of a GWWS server on a destination endpoint. SALT supports the WS-ReliableMessaging Policy Assertion specification to ensure the interoperability with Oracle WebLogic 9.x / 10.

For more information about configuring a reliable GWWS server, see "Configuring Advanced Web Service Messaging Features in the *Oracle SALT Administration Guide*.

WS-RM Policy Assertion Format

Figure C-1 shows a graphical representation of the WS-ReliableMessaging Policy Assertion format in a WS-Policy file.

Figure C-1 WS-ReliableMessaging Policy Assertion Format



WS-RM Assertion File Example

Listing C-1 shows a sample WS-Policy file that contains WS-RM policy assertion.

Listing C-1 Sample WS-ReliableMessaging Policy Assertion File

```
<?xml version="1.0"?>
<wsp:Policy wsp:Name="ReliableSomeServicePolicy"
  xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:beapolicy="http://www.bea.com/wsrm/policy">
  <wsrm:RMAssertion>
    <wsrm:InactivityTimeout Milliseconds="600000" />
    <wsrm:BaseRetransmissionInterval Milliseconds="500"/>
```

```
    <wsrm:ExponentialBackoff />
    <wsrm:AcknowledgementInterval Milliseconds="2000" />
    <beapolicy:Expires Expires="P1D" />
    <beapolicy:QOS QOS="ExactlyOnce InOrder" />
  </wsrm:RMAssertion>
</wsp:Policy>
```

WS-RM Assertion Element Description

All RM assertions are optional, and if not specified, the default value are used. The following definitions describe the RM assertion options.

<wsrm:InactivityTimeout>

Specifies the number of milliseconds, specified with the Milliseconds attribute, which defines an inactivity interval. After time has elapsed, if the destination endpoint has not received a message from the source endpoint, the destination endpoint may terminate current sequence due to inactivity. The source endpoint can also use this parameter.

Sequences never time out by default.

<wsrm:AcknowledgementInterval>

Specifies the maximum interval, in milliseconds, in which the destination endpoint must transmit a stand-alone acknowledgement.

This element is optional. If this element is not specified, There is no time limit by default.

<wsrm:BaseRetransmissionInterval>

Specifies the interval, in milliseconds, that the source endpoint waits after transmitting a message and before it retransmits the message if it receives no acknowledgment for that message. This value will apply to the GWWS server when it sends a response in an outbound sequence.

The default value is 20000 milliseconds.

<wsrm:ExponentialBackoff>

Specifies that the retransmission interval is adjusted using the exponential back off algorithm. This value applies to the GWWS server when it sends a response in an outbound sequence.

<beapolicy:Expires>

Specifies the amount of time after which the reliable Web service expires and does not accept any new sequence messages.

This element has a single attribute, Expires, whose data type is an XML Schema duration type. For example, if you want to set the expiration time to one day, use the following:

```
< beapolicy:Expires Expires="P1D" />
```

The default value is never expire.

<beapolicy:QOS>

Specifies the delivery assurance. SALT supports the following assurances:

- AtMostOnce - Messages are delivered at most once, without duplication. There is possibility that some messages may not be delivered.
- AtLeastOnce - Every message is delivered at least once. There is possibility that some messages are delivered more than once.
- ExactlyOnce - Each message is delivered exactly once, without duplication.
- InOrder - Messages are delivered in the order that they were sent. This delivery assurance can be combined with one of the preceding three assurances.

The default value is "ExactlyOnce InOrder".

<wsrm:RMAssertion>

Main WS-RM assertion that groups all the other assertions under a single element.

The presence of this assertion in a WS-Policy file indicates that the corresponding Web Service application must be invoked reliably.

Oracle SALT WS-SecurityPolicy Assertion 1.2 Reference

The following sections provide SALT WSSP1.2 reference information:

Overview

Oracle SALT implements part of WS-Security protocol version 1.1 for inbound services. Authentication with UsernameToken and X509v3Token are supported. To describe how the authentication is carried out, WS-SecurityPolicy is used in WSDL definition.

In order to communicate with Oracle WebLogic Release 10 via WS-Security 1.1, SALT implements the counterparts of WS-SecurityPolicy (WSSP) 1.2 supported by WebLogic 10. But the supported WSSP 1.2 assertions are limited as follows:

- Protection Assertions
  - Integrity Assertion
    - <sp:SignedParts> Assertion (Limited support)
- Token Assertions:
  - <sp:UsernameToken> Assertion (Limited support)

  - <sp:X509Token> Assertion (Limited support)

- Security Binding Assertions:
  - AsysmmetricBinding Assertion (Limited support)
  - <sp:TransportBinding > Assertion (Limited support)
- Supporting Tokens Assertions:
  - SupportingTokens Assertion (Limited support)

For more details about limitations of WS-SecurityPolicy 1.2 assertions, please refer to SALT WSSP1.2 Assertion Description.

For more information about WSSP 1.2 assertions supported by WebLogic 10, please refer to "Using WS-SecurityPolicy 1.2 Policy Files in the *Oracle WebLogic Web Services Documentation.*

In this document, XML namespace prefix "sp" stands for namespace URI "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512".

SALT WSSP 1.2 Policy File Example

Listing D-1 demonstrates how to apply Username token authentication with WSSP 1.2 assertions.

Listing D-1 WSSP 1.2 Policy File Sample

```
<!-Binding Policy -->
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512">
  <sp:TransportBinding>
    <wsp:Policy>
      <sp:TransportToken>
        <wsp:Policy>
          <sp:HttpToken/>
        </wsp:Policy>
      </sp:TransportToken>
      <sp:AlgorithmSuite>
        <wsp:Policy>
```

```
          <sp:Basic256/>
        </wsp:Policy>
      </sp:AlgorithmSuite>
      <sp:Layout>
        <wsp:Policy>
          <sp:Lax/>
        </wsp:Policy>
      </sp:Layout>
      <sp:IncludeTimestamp/>
    </wsp:Policy>
  </sp:TransportBinding>
  <sp:SupportingTokens>
    <wsp:Policy>
      <sp:UsernameToken
          sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypol
icy/200512/IncludeToken/AlwaysToRecipient">
        <wsp:Policy>
          <sp:WssUsernameToken10/>
        </wsp:Policy>
      </sp:UsernameToken>
    </wsp:Policy>
  </sp:SupportingTokens>
</wsp:Policy>
```

SALT WSSP 1.2 Policy Templates

Oracle SALT provides a number of WS-SecurityPolicy 1.2 template files you can use for most typical Web Service applications. These policy files are located in directory `TUXDIR/udataobj/salt/policy`.

Table D-1 SALT WSSP 1.2 Policy Template Files

Policy File	Description
`wssp1.2-UsernameToken-plain-auth.xml`	Username token with plain text password is sent in the request for authentication.
`wssp1.2-x509v3-auth.xml`	X509 V3 binary token (certificate) is sent in the request for authentication. The request is optionally signed with some message parts in the requests.
`wssp1.2-signbody.xml`	The entire SOAP body is signed.

These template files can be referenced directly in the WSDF files with location value format:

```
salt:<template_file_name>
```

For example, if you want to configure signbody, you can specify the followings in your WSDF file:

```
<Policy location="salt:wssp1.2-signbody.xml" />
```

SALT WSSP1.2 Assertion Description

Below are all Oracle SALT supported WSSP 1.2 assertions and limitations for each one. Customers should obey the limitation when writing their own customized WSSP 1.2 policy files. Oracle SALT does not check any customized WSSP 1.2 policy file against the limitation rules. If something claimed in the customized WSSP 1.2 policy file cannot be supported by Oracle SALT, web service client program may result run time errors.

WS-SecurityPolicy 1.2 assertions not listed below are definitely not supported by Oracle SALT.

<sp:SignedParts>

Specifies the parts of a SOAP message to be digitally signed. Oracle SALT only supports the entire SOAP body to be signed.

Limitations

- Child element <sp:Body> is supported for configuring the entire SOAP body to be signed.
- Child element <sp:Header> is not yet supported.

- No nesting WSSP 1.2 assertion for this assertion.

<sp:UsernameToken>

Specifies username token to be included in the SOAP message. Oracle SALT only supports username token with clear text password defined in WS-Security Username Token Profile 1.0. <UsernameToken> assertion must be used as a nested assertion of Security Binding Assertions and Supporting Token Assertions.

Limitations

- Supported Nesting Assertions
  - <sp:WssUsernameToken10>
- Not yet supported Nesting Assertions
  - <sp:WssUsernameToken11>
  - <sp:NoPassword>
  - <sp:HashPassword>

<sp:X509Token>

Specifies a binary security token carrying an X509 token to be included in the SOAP message. <X509Token> assertion must be used as a nested assertion of Security Binding Assertions and Supporting Token Assertions.

Limitations

- Supported Nesting Assertions
  - <sp:WssX509V3Token10>
  - <sp:WssX509V3Token11>
- Non-Supported Nesting Assertions
  - <sp:WssX509Pkcs7Token10>
  - <sp:WssX509Pkcs7Token11>
  - <sp:WssX509PkiPathV1Token10>
  - <sp:WssX509PkiPathV1Token11>

  - <sp:WssX509V1Token10>
  - <sp:WssX509V1Token11>

<sp:AlgorithmSuite>

Specifies the algorithm suite to be used for performing cryptographic operations with security tokens. <AlgorithmSuite> Assertion must be used as a nested assertion of Security Binding Assertions.

Limitations

- Supported Nesting Algorithm Suite
  - <sp:Basic256>
- Non-Supported Nesting Algorithm Suites
  - All the other Algorithm Suite listed in the WS-Security Policy 1.2 specification.

<sp:Layout>

Specifies the layout rules when adding items to the security header. <Layout> Assertion must be used as a nested assertion of Security Binding Assertions.

Limitations

- Supported Nesting Layout rules
  - <sp:Lax>
- Non-Supported Nesting Layout rules
  - <sp:Strict>
  - <sp:LaxTimestampFirst>
  - <sp:LaxTimestampLast>

<sp:TransportBinding >

Specifies the message protection and security correlation is provided using the means of the transport. The <TransportBinding> token is used mainly for carrying isolated Username Token in the SOAP message.

Limitations

- Supported Nesting Assertions
  - <sp:TransportToken>
  - <sp:AlgorithmSuite>
  - <sp:Layout>
  - <sp:IncludeTimestamp>
- Nesting Assertion <sp:TransportToken> only supports <sp:HttpToken>

Listing D-2 shows an Oracle SALT supported TransportToken Assertion example.

Listing D-2 Supported TransportToken Assertions

```
<sp:TransportBinding>
  <wsp:Policy>
    <sp:TransportToken>
      <wsp:Policy>
        <sp:HttpToken />
      </wsp:Policy>
    </sp:TransportToken>
    <sp:Algorithm>
      <wsp:Policy>
        <sp:Basic256>
      </wsp:Policy>
    </sp:Algorithm>
  </wsp:Policy>
</sp:TransportBinding>
```

<sp:AsymmetricBinding>

Specifies the message protection is provided by means defined in WS-Security SOAP Message Security, and the request and response message can use distinct keys for encryption and signature, because of their different lifecycles. The <AsymmetricBinding> Assertion is used mainly for carrying X.509 binary security token in the SOAP request messages for inbound calls.

Limitations

- Supported Nesting Assertions
  - <sp:InitiatorToken>
  - <sp:RecipientToken>
  - <sp:AlgorithmSuite>
  - <sp:Layout>
  - <sp:IncludeTimestamp>
  - <sp:ProtectTokens>
  - <sp:OnlySignEntireHeadersAndBody>
- Non-supported Nesting Assertions
  - <sp:InitiatorSignatureToken>
  - <sp:InitiatorEncryptToken>
  - <sp:RecipientSignatureToken>
  - <sp:RecipientEncryptToken>
  - <sp:EncryptBeforeSigning>
  - <sp:EncryptSignature>
- <sp:InitiatorToken> must be associated with <sp:X509Token> and the Token inclusion type must be “AlwaysToRecipient“
- <sp:RecipientToken> must be associated with <sp:X509Token> and the Token inclusion type must be “Never”

Listing D-3 shows an Oracle SALT supported AsymmetricBinding assertion example. This assertion indicates the X.509 V3 binary token that defined in WS-Security X.509 Token Profile 1.1 specification is used for digital signature for the SOAP request messages and the X.509 token is always included in the SOAP message security header:

Listing D-3 Supported AsymmetricBinding Assertion

```
<sp:AsymmetricBinding>
  <wsp:Policy>
    <sp:InitiatorToken>
```

```
        <wsp:Policy>
          <sp:X509Token
              sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securit
ypolicy/200512/IncludeToken/AlwaysToRecipient">
            <wsp:Policy>
              <sp:WssX509V3Token11 />
            </wsp:Policy>
          </sp:X509Token>
        </wsp:Policy>
      </sp:InitiatorToken>
      <sp:RecipientToken>
        <wsp:Policy>
          <sp:X509Token
              sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securit
ypolicy/200512/IncludeToken/Never">
            <wsp:Policy>
              <sp:WssX509V3Token11 />
            </wsp:Policy>
          </sp:X509Token>
        </wsp:Policy>
      </sp:RecipientToken>
      <sp:Algorithm>
        <wsp:Policy>
          <sp:Basic256>
        </wsp:Policy>
      </sp:Algorithm>
      <sp:Layout>
        <wsp:Policy>
          <sp:Lax>
        </wsp:Policy>
      </sp:Layout>
      <sp:IncludeTimestamp />
    </wsp:Policy>
  </sp:AsymmetricBinding>
```

<sp:SupportingToken>

Specifies security tokens that are included in the security header and may optionally include additional message parts to sign and/or encrypt. For Oracle SALT, <SupportingToken> Assertion is used mainly to include Username Token in the security header when <sp:AsymmetricBinding> Assertion is used.

Limitations

- Supported Nesting Assertions
  - <sp:UsernameToken>
  - <sp:X509Token>
- Not-non Supported Nesting Assertions
  - <sp:SignedParts>
  - <sp:SignedElements
  - <sp:EncryptedParts>
  - <sp:EncryptedElements>
- All supported token assertions must be defined with Token inclusion type “AlwaysToRecipient”.

Listing D-4 shows an Oracle SALT supported SupportingToken assertion example. This assertion indicates the Username token is always included in SOAP request messages:

Listing D-4 Supported SupportingToken Assertion

```
  <sp:SupportingTokens>
    <wsp:Policy>
      <sp:UsernameToken
          sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypol
icy/200512/IncludeToken/AlwaysToRecipient">
        <wsp:Policy>
          <sp:WssUsernameToken10/>
        </wsp:Policy>
      </sp:UsernameToken>
```

```
  </wsp:Policy>
</sp:SupportingTokens>
```

APPENDIX E

Oracle SALT WS-SecurityPolicy Assertion 1.0 Reference

The following sections provide SALT WS-SecurityPolicy (WSSP) 1.0 assertion reference information:

Overview

Oracle SALT implements part of WS-Security protocol version 1.0 for inbound services. Authentication with UsernameToken and X509v3Token are supported. WS-SecurityPolicy 1.0 assertions are used in WSDL definition to describe how the authentication is carried out. The WS-SecuirtyPolicy1.0 specification (2002) is supported in order to ensure the interoperability with Oracle WebLogic 9.x.

Below are all Oracle SALT supported WS-SecurityPolicy 1.0 assertions:

- SecurityToken Assertions:
  - UsernameToken Assertion and X509Token Assertion
- Integrity Assertion

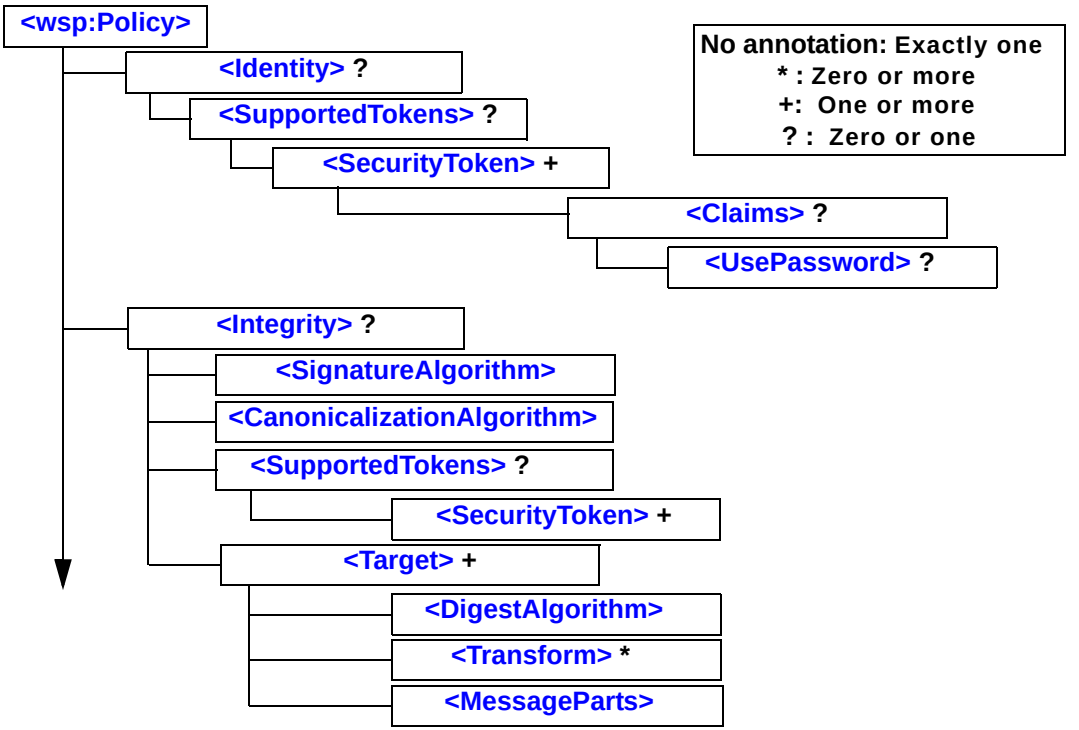
- Identity Assertion

There are some extension assertions used in WebLogic 9.x, SALT only implements a subset of them. Integrity Assertion is only used when using X509v3 token for authentication. And the only message part can be specified for signature is the whole SOAP Body.

SALT WSSP 1.0 Policy Assertion Format

Figure E-1 shows a graphical representation of the Oracle SALT supported WS-SecurityPolicy 1.0 Assertion format in a WS-Policy file.

Figure E-1 SALT Supported WS-SecurityPolicy 1.0 Assertion Format



SALT WSSP 1.0 Assertion File Example

Listing E-1 demonstrates how to apply Username token authentication with WSSP 1.0 Assertions.

Listing E-1 WSSP 1.0 Policy File Sample

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://www.bea.com/WLS/security/policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssec
urity-utility-1.0.xsd">
  <wssp:Identity>
    <wssp:SupportedTokens>
      <wssp:SecurityToken
            TokenType="http://docs.oasis-open.org/wss/2004/01/oasis-200401
-wss-username-token-profile-1.0#UsernameToken">
        <wssp:Claims>
          <wssp:UsePassword>http://docs.oasis-open.org/wss/2004/01/oasis-2
00401-wss-username-token-profile-1.0#PasswordText</wssp:UsePassword>
        </wssp:Claims>
      </wssp:SecurityToken>
    </wssp:SupportedTokens>
  </wssp:Identity>
</wsp:Policy>
```

SALT WSSP 1.0 Policy Templates

Oracle SALT provides a number of WS-SecurityPolicy 1.0 template files you can use for most typical Web Service applications. These policy files are located in directory `TUXDIR/udataobj/salt/policy` as shown in Table E-1.

Table E-1 SALT WSSP 1.0 Policy Template Files

Policy File	Description
wssp1.0-UsernameToken-plain-auth.xml	Username token with plain text password is sent in the request for authentication.
wssp1.0-x509v3-auth.xml	X509 V3 binary token (certificate) is sent in the request for authentication. The request is optionally signed with some message parts in the requests.
wssp1.0-signbody.xml	The whole SOAP body is signed.

These template files can be referenced directly in the WSDF files with location value format:

```
salt:<template_file_name>
```

For instance, if you want to configure signbody, you can specify the followings in your WSDF file:

```
<Policy location="salt:wssp1.0-signbody.xml" />
```

SALT WSSP 1.0 Assertion Element Description

Oracle SALT implements part of WebLogic 9.x / 10 WS-SecurityPolicy 1.0 assertions. For a complete list of WSSP 1.0 assertions supported by WebLogic, see http://edocs.bea.com/wls/docs100/webserv_ref/sec_assert.html

<CanonicalizationAlgorithm>

Specifies the algorithm used to canonicalize the SOAP message elements that are digitally signed. Table E-2 lists the <CanonicalizationAlgorithm> attributes.

Table E-2 <CanonicalizationAlgorithm> Attribute

Attribute	Description	Required?
URI	The algorithm used to canonicalize the SOAP message being signed. SALT supports only the following canonicalization algorithm: `http://www.w3.org/2001/10/xml-exc-cl4n#`	Yes

<Claims>

Specifies additional metadata information that is associated with a particular type of security token. Depending on the type of security token, you must specify the following child elements:

- For username tokens, you must specify a <UsePassword> child element to specify what kind of the password will be used for in username authentication.

This element does not have any attributes.

<DigestAlgorithm>

Specifies the digest algorithm that is used when digitally signing the specified parts of a SOAP message. Use the <MessageParts> sibling element to specify the parts of the SOAP message you want to digitally sign. Table E-3 lists the <DigestAlgorithm> attributes.

Table E-3 <DigestAlgorithm> Attributes

Attribute	Description	Required?
URI	The digest algorithm that is used when digitally signing the specified parts of a SOAP message. SALT supports only the following digest algorithm: `http://www.w3.org/2000/09/xmldsig#sha1`	Yes

<Identity>

Specifies the type of security tokens (username or X.509) that are supported for authentication.

This element has no attributes.

<Integrity>

Specifies that part or all of the SOAP message must be digitally signed, as well as the algorithms and keys that are used to sign the SOAP message.

For example, a Web Service may require that the entire body of the SOAP message must be digitally signed and only algorithms using SHA1 and an RSA key are accepted. Table E-4 lists the <Integrity> attributes.

Table E-4 <Integrity> Attributes

Attribute	Description	Required?
SignToken	Specifies whether the security token, specified using the <SecurityToken> child element of <Integrity>, should also be digitally signed, in addition to the specified parts of the SOAP message. The valid values for this attribute are true and false. The default values is true.	No

<MessageParts>

Specifies the parts of the SOAP message that should be signed. SALT only supports certain pre-defined message part function, wsp:Body(), i.e. the entire SOAP body to be digitally signed.

The MessageParts assertion is always a child of a <Target> assertion. The <Target> assertion can be a child of an Integrity assertion (to specify how the SOAP message is digitally signed).

See "Usage of MessageParts" for more information about how to specify the parts of the SOAP message that should be signed. Table E-5 lists the <MessageParts> attributes.

Table E-5 <MessageParts> Attributes

Attribute	Description	Required?
Dialect	Identifies the dialect used to identity the parts of the SOAP message that should be signed. SALT only supports the following value: • `http://schemas.xmlsoap.org/2002/12/wsse#part` Convenience dialect used to specify parts of SOAP message that should be signed.	Yes

<SecurityToken>

Specifies the security token that is supported for authentication or digital signatures, depending on the parent element.

If this element is defined in the <Identity> parent element, then is specifies that a client application, when invoking the Web Service, must attach a security token to the SOAP request.

For example, a Web Service might require that the client application present a Username token for the Web Service to be able to access Tuxedo service. If this element is part of <Integrity>, then it specifies the token used for digital signature.

The specific type of the security token is determined by the value of its TokenType attribute, as well as its parent element. Table E-6 lists the <SecurityToken> attributes.

Table E-6 <SecurityToken> Attributes

Attribute	Description	Required?
IncludeInMessage	Specifies whether to include the token in the SOAP message. Valid values are true or false. The default value of this attribute is true when used in the <Integrity> assertion. The value of this attribute is always true when used in the <Identity> assertion, even if you explicitly set it to false.	No
TokenType	Specifies the type of security token. Valid values are: • `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3` (To specify a binary X.509 v3 token) • `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken` (To specify a username token)	Yes

<SignatureAlgorithm>

Specifies the cryptographic algorithm used to compute the digital signature. Table E-7 lists the <SignatureAlgorithm> attributes.

Table E-7 <SignatureAlgorithm> Attributes

Attribute	Description	Required?
URI	Specifies the cryptographic algorithm used to compute the signature. **Note:** Be sure that you specify an algorithm that is compatible with the certificates you are using in your enterprise. Valid values are: • `http://www.w3.org/2000/09/xmldsig#rsa-sha1` • `http://www.w3.org/2000/09/xmldsig#dsa-sha1`	Yes

<SupportedTokens>

Specifies the list of supported security tokens that can be used for authentication, or digital signatures, depending on the parent element.

This element has no attributes.

<Target>

Encapsulates information about which targets of a SOAP message are to be signed. When used in <Integrity>, you can specify the <DigestAlgorithm>, <Transform>, and <MessageParts> child elements.

Ideally, you can have one or more targets. But at most one target is enough for SALT, since SALT only supports the entire SOAP body to be configured for digital signature.

This element has no attributes.

<Transform>

Specifies the URI of a transformation algorithm that is applied to the parts of the SOAP message that are signed. Only can exist in a child element of the <Integrity> element.

You can specify zero or more transforms, which are executed in the order they appear in the <Target> parent element. Table E-8 lists the <Transform> attributes.

Table E-8 <Transform> Attributes

Attribute	Description	Required?
URI	Specifies the URI of the transformation algorithm. SALT only supports the following transformation algorithm: • `http://www.w3.org/2000/09/xmldsig#base64` (Base64 decoding transforms) For detailed information about these transform algorithms, see XML-Signature Syntax and Processing.	Yes

<UsePassword>

Specifies that whether the plaintext or the digest of the password appear in the SOAP messages. This element is used only with username tokens. In SALT, it must be specified as plaintext. Table E-9 lists the <UsePassword> attributes.

Table E-9 <UsePassword> Attributes

Attribute	Description	Required?
Type	Specifies the type of password. SALT only supports cleartext passwords, the value URI is: • `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText` Specifies that cleartext passwords should be used in the SOAP messages. **Note:** For backward compatibility reasons, the preceding URI can also be specified with an initial "www." For example: – `http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText`	Yes

Usage of MessageParts

When you use the <Integrity> assertion in your WS-Policy file, you are required to also use the Target child assertion to specify the targets of the SOAP message to digitally sign. The <Target>

assertion in turn requires that you use the <MessageParts> child assertion to specify the actual parts of the SOAP message that should be digitally signed. You can use the `Dialect` attribute of <MessageParts> to specify the dialect used to identify the SOAP message parts. Oracle SALT Web services security module supports only the following dialect:

- Pre-Defined Message Part Selection Function

Be sure that you specify a message part that actually exists in the SOAP messages that result from a client invoke of a message-secured Web Service. If the Web Services security module encounters an inbound SOAP message that does not include a part that the WS-Policy file indicates should be signed or encrypted, then the Web Services security module returns an error and the invoke fails.

Pre-Defined Message Part Selection Function

This section shows SALT supported functions (shown in Table E-10) that are used with the "`http://schemas.xmlsoap.org/2002/12/wsse#part`" dialect for selecting parts of a message:

Table E-10 SALT Supported Message Part Selection Function

Function	Description
wsp:Body()	Specifies the entire SOAP message body to be selected as one part

You can only specify the entire SOAP body to be signed. It is recommended that you use the dialect that pre-defines the `wsp:Body()` function for this purpose.

Listing E-2 shows a `wsp:Body()` function example

Listing E-2 wsp:Body() Function

```
<wssp:MessageParts
    Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
      wsp:Body()
</wssp:MessageParts>
```

Oracle SALT SCA ATMI Binding Reference

The following sections provide SCA ATMI Binding reference information:

SCA ATMI Binding Schema

Listing F-1 shows how the ATMI binding element (`<binding.atmi>`) is defined. This is a pseudoschema that depicts how the grammar is used and what parameters are legal.

Notes: The parameters "`transactionalintent legacyintent`" are not literal values. `transactionalintent` can be substituted with "`suspendsTransaction`" or "`propagatesTransaction`" or omitted. "`legacyintent`" can be substituted with "legacy" or omitted.

Parameters with a `?` may be specified 0 or 1 times, and parameters with * may be specified 0 or more times.

When using the `<binding.atmi>`element, the total length of `/reference/@name (or/service/@name)` and method name must be equal to or less than the maximum length of a Tuxedo service name (this varies depending on the Tuxedo release). To overcome this limitation, see `</binding.atmi/map>`.

Listing F-1 SCA ATMI Binding Pseudoschema

```
<binding.atmi requires="transactionalintent legacyintent"?>
       <tuxconfig>...</tuxconfig>?

       <map target="name">...</map>*
       <serviceType target="name">...</serviceType>*
       <inputBufferType target="name">...</inputBufferType>*
       <outputBufferType target="name">...</outputBufferType>*
       <errorBufferType target="name">...</errorBufferType>*
       <workStationParameters>?
              <networkAddress>...</networkAddress>?
              <secPrincipalName>...</secPrincipalName>?
              <secPrincipalLocation>...</secPrincipalLocation>?
              <secPrincipalPassId>...</secPrincipalPassId>?
              <encryptBits>...</encryptBits>?
       </workStationParameters>
       <authentication>?
              <userName>...</userName>?
              <clientName>...</clientName>?
              <groupName>...</groupName>?
              <passwordIdentifier>...</passwordIdentifier>?
              <userPasswordIdentifier>...
                                          </userPasswordIdentifier>?
       </authentication>
       <fieldTablesLocation>...</fieldTablesLocation>?
       <fieldTables>...</fieldTables>?
       <fieldTablesLocation32>...</fieldTablesLocation32>?
       <fieldTables32>...</fieldTables32>?
       <viewFilesLocation>...</viewFilesLocation>?
       <viewFiles>...</viewFiles>?
       <viewFilesLocation32>...</viewFilesLocation32>?
       <viewFiles32>...</viewFiles32>?
       <remoteAccess>...</remoteAccess>?
       <transaction timeout="xsd:long"/>?
</binding.atmi>
```

SCA ATMI Binding Attributes Description

The <binding.atmi> element supports the following attributes

- </binding.atmi/@requires>
- </binding.atmi/tuxconfig>
- </binding.atmi/map>
- </binding.atmi/serviceType>
- </binding.atmi/inputBufferType>, </binding.atmi/outputBufferType>, </binding.atmi/errorBufferType>
- </binding.atmi/workStationParameters>
- </binding.atmi/authentication>
- </binding.atmi/fieldTablesLocation>
- </binding.atmi/fieldTablesLocation32>
- </binding.atmi/fieldTables>
- </binding.atmi/fieldTables32>
- </binding.atmi/viewFilesLocation>
- </binding.atmi/viewFilesLocation32>
- </binding.atmi/viewFiles>
- </binding.atmi/viewFiles32>
- </binding.atmi/remoteAccess>
- </binding.atmi/transaction/@timeout>

</binding.atmi/@requires>

- When this attribute contains the `legacy` value, it is used to perform interoperability with existing Tuxedo services. When not specified, communications are assumed to have SCA to SCA semantics where the actual Tuxedo service name is constructed from `/service/@name` or `/reference/@name` and actual method name (see Listing F-1), unless a `/binding.atmi/map` element is defined. When this attribute encounters a legacy

value, and no /binding.atmi/map element is defined for the method being called, it has the following run-time behavior:

  – In a <reference> element: the value specified in the /reference/@name is used to perform the Tuxedo call, with semantics used according to the interface method.

  – In a <service> element: the Tuxedo service specified in the /binding.atmi/map element is advertised, and mapped to the method specified in the /binding.atmi/map/@target attribute.

- When this attribute contains a transaction value, it specifies the transactional behavior that the binding extension follows when this binding is used. Possible values are as follows:

  – not specified (no value) - all transactional behavior is controlled by the Tuxedo configuration. If the Tuxedo configuration supports transactions, then one may be propagated if it exists. If the Tuxedo configuration does not support transactions and one exists then an error will occur. However, a transaction cannot start if one does not already exist.

  – suspendsTransaction - transaction context is propagated to the called service. For a <service> element when a transaction is present, it is automatically suspended before invoking the application code. It resumes afterwards, regardless of the outcome of the invocation. For a <reference> element, it is equivalent to making a tpcall() with the TPNOTRAN flag.

  – propagatesTransaction - only applicable to <reference> elements. It is ignored for <service> elements. This value starts a new transaction if one does not already exist, otherwise it participates in the existing transaction.

    Such behavior can be obtained in a component or composite <service> element by configuring AUTOTRAN in the UBBCONFIG file. An error is generated if a Tuxedo server hosts the SCA component implementation and it is not configured in a transactional group in the UBBCONFIG file.

</binding.atmi/tuxconfig>

Used in <reference> elements when /binding.atmi/workstationParameters is not set, and for client-only processes. It indicates the Tuxedo application that the process should join. One process can join multiple applications, or switch applications without having to restart.

If not set, the TUXCONFIG environment variable is used. If not set, but one is required, the process exits and returns an error.

</binding.atmi/map>

For <reference> elements, </binding.atmi/map>provides the Tuxedo service name that should be used when performing the invocation to the corresponding /binding.atmi/map/@target value, this value being the name of the method being called.

For <service> elements, </binding.atmi/map> provides the Tuxedo service name that should be advertised for the corresponding /binding.atmi/map/@target value.

The /binding.atmi/map/@target value *must* match the method name of the corresponding service interface.

If a /binding.atmi/map element is present, it takes precedence over any other form of service/method to Tuxedo service name mapping. See </binding.atmi/@requires> attribute.

</binding.atmi/serviceType>

Optional element that specifies the type of call being handled. The accepted values are:

- Oneway - the call will not expect a response.
- RequestResponse - regular call paradigm, default value.

</binding.atmi/inputBufferType>, </binding.atmi/outputBufferType>, </binding.atmi/errorBufferType>

Optional elements that specify the type of buffer that the processes exchange. The inputBufferType element is used by the binding extension to determine or check the type of the request.

The outputBufferType element is used by the binding extension to determine or check the type of the reply.

The errorBufferType element is used to determine the type of buffer specified in the data portion of the Exception thrown received by a client or thrown by a server.

Table F-1 lists supported values and corresponding Tuxedo buffer types. An incorrect value or syntax is detected at run time and causes the call to fail. If not specified, the default value used is STRING.

Table F-1 SCA Supported Tuxedo Buffer Types

/binding.atmi/bufferType value	Tuxedo buffer type	Note
STRING	STRING	
CARRAY	CARRAY	
X_OCTET	X_OCTET	
VIEW	VIEW	Format is VIEW/<subtype>
X_C_TYPE	X_C_TYPE	Format is X_C_TYPE/<subtype>
X_COMMON	X_COMMON	Format is: X_COMMON/<subtype>
VIEW32	VIEW32	Format is VIEW32/<subtype>
XML	XML	
FML	FML	Format is: FML/<subtype>, <subtype> is optional The <subtype> value allows to specify the SDO type to use for that message (request or response) when it is described in an XML schema **Note:** FML32 <subtype> is not available for JATMI binding.

Table F-1 SCA Supported Tuxedo Buffer Types

/binding.atmi/bufferType value	Tuxedo buffer type	Note
FML32	FML32	Format is: FML32/<subtype>, <subtype> is optional The <subtype> value allows to specify the SDO type to use for that message (request or response) when it is described in an XML schema **Note:** FML32 <subtype> is not available for JATMI binding.
MBSTRING	MBSTRING	

</binding.atmi/workStationParameters>

An optional element that specifies parameters specific to the Tuxedo WorksStation protocol. Only used in references.

- /binding.atmi/workStationParameters/networkAddress

  The address of the workstation listener to which this application will connect. Any address format accepted by the Tuxedo workstation software is allowed. The most common address format is:

  //<hostname or IP address>:<port>.

  For more information, see the SALT Programming Guide

More than one address can be specified (if required), by specifying a comma-separated list of pathnames for WSNADDR Addresses are tried in order until a connection is established. Any member of an address list can be specified as a parenthesized grouping of pipe-separated network addresses. For example:

```
<networkAddress>
(//m1.acme.com:3050|//m2.acme.com:3050),//m3.acme.com:3050
</networkAddress>
```

Tuxedo randomly selects one of the parenthesized addresses. This strategy distributes the load randomly across a set of listener processes. Addresses are tried in order until a connection is established.

On versions of Tuxedo that support ipv6, the corresponding addressing format will also be supported, following the same format as used in `WSNADDR` for Tuxedo /WS clients.

- `secPrincipalName`, `secPrincipalLocation`, `secPrincipalPassId`

  These parameters specify the necessary parameters when an SSL connection is required by a workstation client. The password is stored in a separate file and accessed using a callback mechanism. The default callback uses the `password.store` file maintained using the `scapasswordtool` command. For more information, see the SALT Programming Guide

- `encryptBits`

  Specifies the encryption strength that this client connection will attempt to negotiate. The format is `<minencryptbits>/<maxencprytbits>` (for example, 128/128), those values being numerical. Invalid values will result in a configuration exception being thrown. Values can be 0 (if no encryption is used), or 40, 56, 128, or 256 (if the number specified is the number of significant bits in the encryption key).

</binding.atmi/authentication>

Specifies the security parameters used in reference-type calls to establish a connection with the Tuxedo application. The following values respectively correspond to the TPINFO structure elements `usrname`, `cltname`, `grpname` and `passwd` (for more information, see `tpinit(3c)` in the Oracle Tuxedo ATMI C Function Reference guide):

- `/binding.atmi/authentication/userName`
- `/binding.atmi/authentication/clientName`
- `/binding.atmi/authentication/groupName`
- `/binding.atmi/authentication/passwordIdentifier-(application password)`
- `/binding.atmi/authentication/userPasswordIdentifier-(user password in per-user authentication)`

  Passwords are not stored in clear text, but are looked up using an identifier. A callback function may be used to retrieve passwords. For more information, see `setSCAPasswordCallback()`in the Oracle SALT Reference Guide.

  By default, passwords are maintained encrypted in a passwords store file located in the same directory as the composite file that contains the `/reference/binding.atmi/authentication/passwordIdentifier` or

`/reference/binding.atmi/authentication/userPasswordIdentifier` element. This identifier is read as necessary to perform authentication.

For more information, see `scapasswordtool` and setSCAPasswordCallback(3c) in the Oracle SALT Reference Guide.

Note: This information should be handled with policy sets and intents when the SCA Kernel supports it.

</binding.atmi/fieldTablesLocation>

Optional element that specifies a directory in the local file system where field tables should be searched. If a relative path is specified, files are searched in that location relative to `$APPDIR`, otherwise the location is assumed to be absolute.

</binding.atmi/fieldTablesLocation32>

Same as `fieldTablesLocation`, but for FML32 buffers.

</binding.atmi/fieldTables>

Optional element that specifies the FML field tables available. Field tables are searched in the location specified by the `/binding.atmi/fieldTablesLocation element`.

If the `/binding.atmi/bufferType` value is FML and this element is not specified or invalid (that is, the tables indicated cannot be found or are not field tables), an error is displayed at initialization time for client processes, or boot time for server processes.

</binding.atmi/fieldTables32>

Same as `fieldTables`, but for FML32 buffers.

</binding.atmi/viewFilesLocation>

Optional element that specifies a directory in the local file system where view tables should be searched. If a relative path is specified, files are searched in that location relative to `$APPDIR`, otherwise the location is assumed to be relative.

</binding.atmi/viewFilesLocation32>

Same as `viewTablesLocation`, but for VIEW32 buffers.

</binding.atmi/viewFiles>

Optional element that specifies the VIEW files to be used by the affected component(s). If the `/binding.atmi/bufferType` value is VIEW and this element is not specified or invalid (that is, the files indicated cannot be found, or are not view files), an error is displayed at run time for client processes, or boot time for server processes.

</binding.atmi/viewFiles32>

Same as ViewFiles but for VIEW32 buffers.

Note: FML/FML32 and VIEW/VIEW32 parameters are optional and may be omitted, in which case the corresponding Tuxedo environment variables are required (FLDTBLDIR/32, FLDTBLS/32, VIEWDIR/32 and VIEWFILES/32). If neither are used, an error message is printed at run time when attempting to use a fielded buffer. If both are set, the parameters contained in the SCDL code take precedence.

</binding.atmi/remoteAccess>

Optional element that specifies the communication protocol with one of the values below. The default is Native.

- Native - indicates that components use standard Tuxedo native communications (IPC queues)
- WorkStation - indicates that components use the Tuxedo /WS communication protocol.

  If set to this value, the binding extension checks that the `/binding.atmi/workStationParameters` element is also populated and valid; if not, it reports a run-time error message.

</binding.atmi/transaction/@timeout>

Specifies the amount of time, in seconds, a transaction can execute before timing out. This attribute affects components or clients that effectively start a global transaction. It is mandatory for `<reference>` components and ignored if set on `<service>` components. Additionally, the value is ignored on components for which the transaction has already been started. If a transaction needs to be started and this attribute is not present (for example, "`requires=propagatesTransaction`" is set), a configuration error occurs.

See Also

- `buildscaclient`, `buildscacomponent`, and buildscaserver in the Oracle SALT Command Reference.
- Oracle SALT Programming Guide