BEA SALT™

Reference Guide

Version 2.0
Document Revised: August 31, 2007

Contents

BEA SALT Command Reference

SALT Web Service Definition File Reference

SALT Deployment File Reference

BEA SALT WS-ReliableMessaging Policy Assertion Reference

BEA SALT WS-SecurityPolicy Assertion 1.2 Reference

SALT WS-SecurityPolicy Assertion 1.0 Reference

BEA SALT Command Reference

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

Table 1 BEA SALT System Processes and Commands

Name	Description
Introduction to Tables and Files	Overview of this document
GWWS	
tmwsdlgen	
wsloadcf	
wsdlcvt	
wsadmin	

GWWS

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 BEA 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 BEA SALT Deployment File.

Environment Variables

The environment variable SALTCONFIG must be set before GWWS server is booted.

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: In 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

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 BEA 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 BEA 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 runtime 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

wsloadcf

Name

wsloadcf – Read a SALT Deployment file and other referenced artifacts, and 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 BEA 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 runtime 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

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 BEA 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 BEA SALT product.

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

`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 following suffixes are appended after the basename:

Table 2 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

wsadmin

Name

`wsadmin` – BEA 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 BEA 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 runtime 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 runtime 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
```

2.The following command inspects runtime 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

APPENDIX A

SALT Web Service Definition File Reference

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

Overview

The BEA SALT Web Service Definition File (WSDF) is an XML-based file used to define BEA 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)

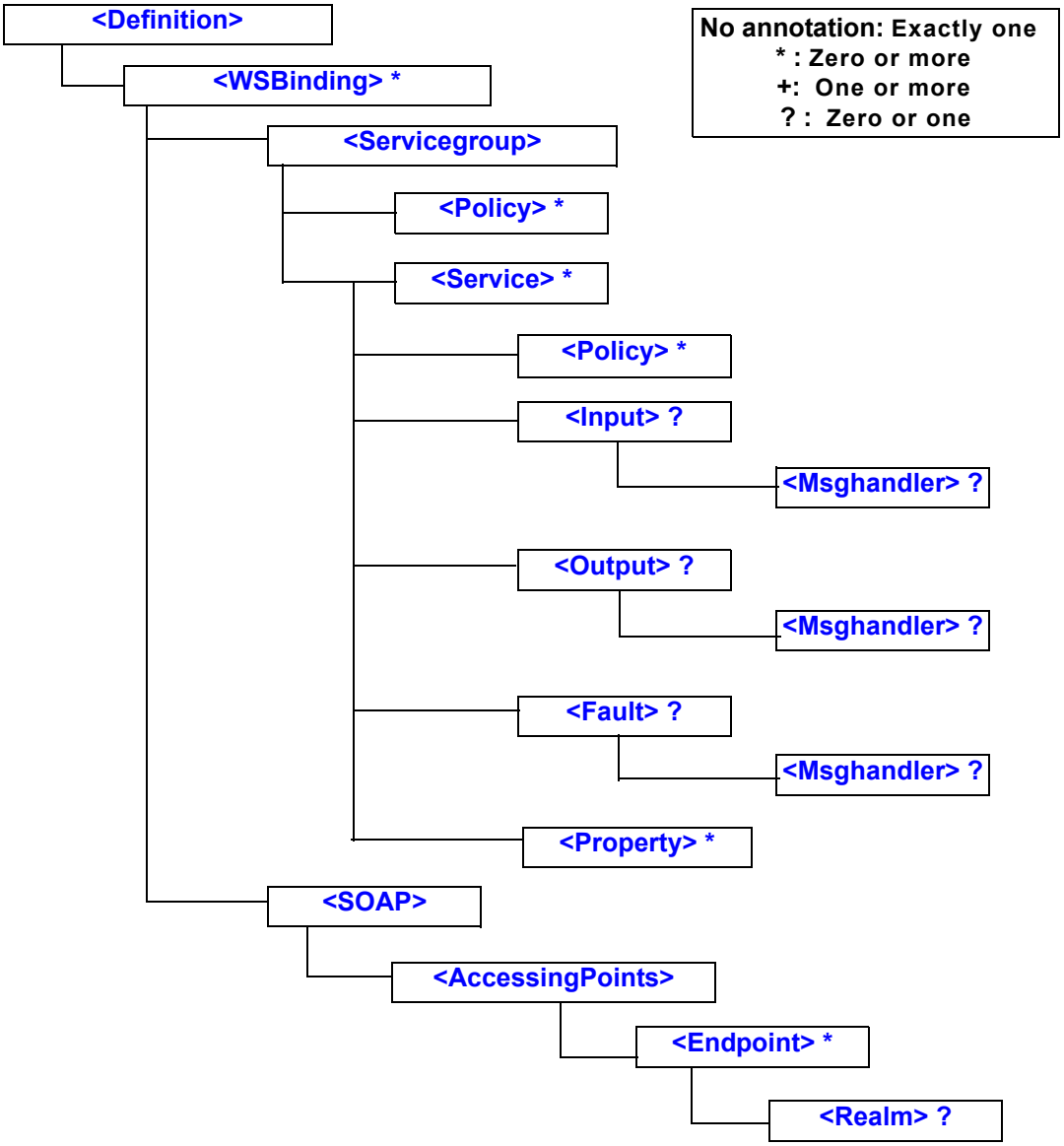
  A non-native `WSDF` is generated from an external WSDL file via the SALT WSDL converter (`wsdlcvt`). In most cases, you do not need to change the generated WSDF except for configuring advanced features.

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

BEA 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.

BEA 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>
```

BEA SALT WSDF Element Descriptions

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

<Definition>

The WSDF file root element.

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 <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 <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 <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 <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 <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 <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 *BEA 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 <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, BEA 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"/> BEA 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 *BEA SALT Administration Guide* • "Configuring Message-Level Web Service Security" in the *BEA SALT Administration Guide* • BEA SALT WS-ReliableMessaging Policy Assertion Reference • BEA SALT WS-SecurityPolicy Assertion 1.2 Reference • SALT WS-SecurityPolicy Assertion 1.0 Reference	No

<Property>

Specifies SALT specific properties for each service object.

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

The following table 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 <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 is no attribute for this element.

<Endpoint>

Specifies each accessing endpoint for the WSBinding object.

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 *BEA SALT Administration Guide*.

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

APPENDIX B

SALT Deployment File Reference

The following sections provide SALT Deployment File reference information

Overview

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

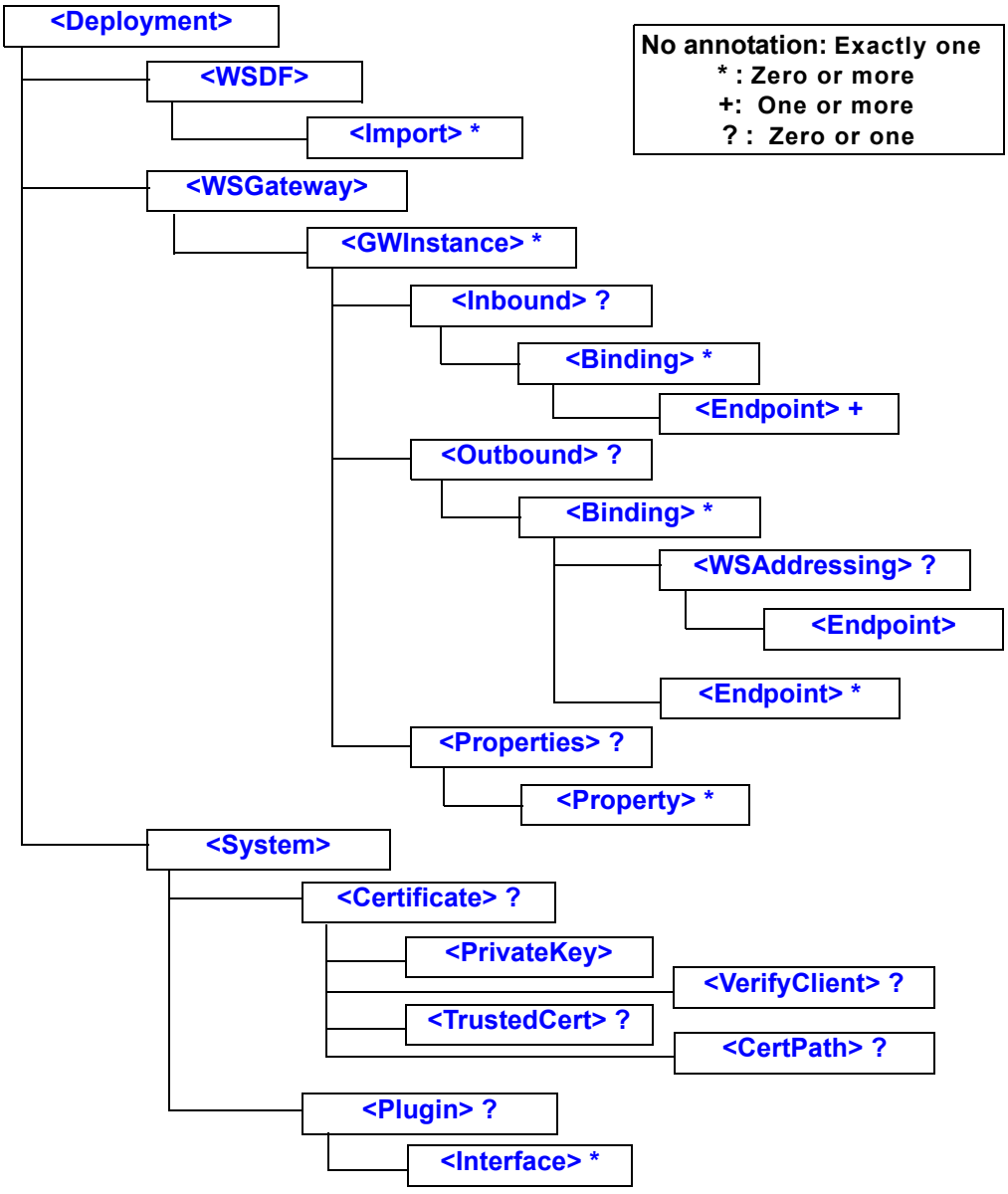
- lists all necessary Web Service Definition Files (WSDF)
- specifies how many `GWWS` servers are deployed on a Tuxedo machine
- associates inbound and outbound Web Service access endpoints for each `GWWS` server.

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

BEA SALT SALTDEPLOY Format

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

Figure B-1 SALT Deployment File Format



XML Schema

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

BEA 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>
```

BEA 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 <Import> Attributes

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

<WSGateway>

Top element that encapsulates all GWWS instance definitions.

There is no attribute for this element.

<GWInstance>

Specifies a single GWWS instance.

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 is no attribute for this element.

<Binding>

Specifies a concrete WSBinding object as either an inbound or outbound binding, depending on the parent element.

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.

<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-4 <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-5 <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 is no attribute for this element.

<Property>

Specify one `GWWS` property.

Table B-6 <Property> Attributes

Attribute	Description	Required
`name`	Specifies the property name. Table B-7 lists all the GWWS server properties.	Yes
`value`	Specifies the property value.	Yes

Table B-7 GWWS <Property> List

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`

Table B-7 GWWS <Property> List

Property	Description	Values
`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`
`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 is no attribute 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-8 <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 BEA SALT Plug-ins" in the *BEA SALT Programming Web Services*.

APPENDIX C

BEA SALT WS-ReliableMessaging Policy Assertion Reference

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

Overview

BEA 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.

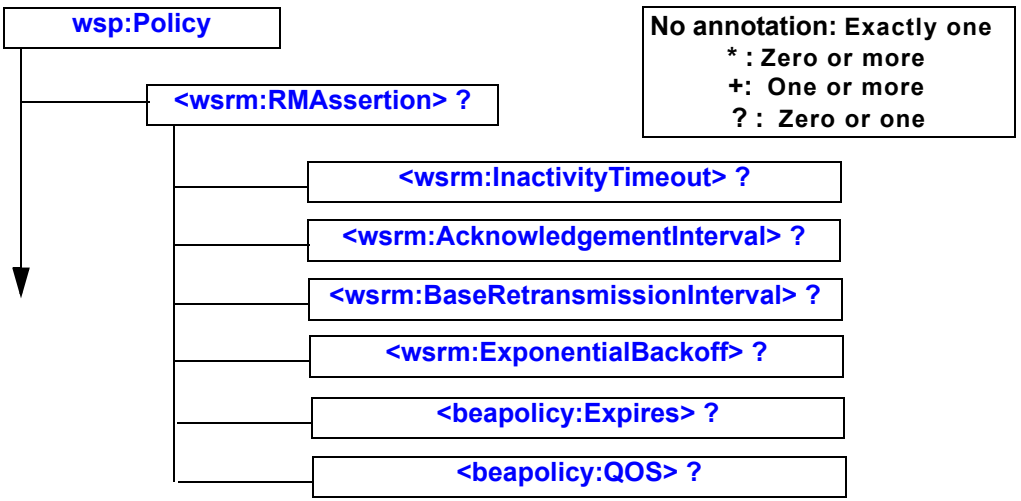
A WS-Policy file containing WS-ReliableMessaging Policy Assertion is used to configure the reliable messaging capabilities of a GWWS server on a destination endpoint. SALT supports the WS-ReliableMessaging Policy Assertion specification to ensure the interoperability with BEA WebLogic 9.x / 10.

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

BEA SALT WS-SecurityPolicy Assertion 1.2 Reference

The following sections provide SALT WSSP1.2 reference information

Overview

BEA 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 BEA 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 *BEA 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

BEA 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 BEA 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. BEA 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 BEA SALT, web service client program may result runtime errors.

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

<sp:SignedParts>

Specifies the parts of a SOAP message to be digitally signed. BEA 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. BEA 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 a BEA 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 a BEA 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 BEA 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 a BEA 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

SALT WS-SecurityPolicy Assertion 1.0 Reference

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

Overview

BEA 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 BEA WebLogic 9.x.

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

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

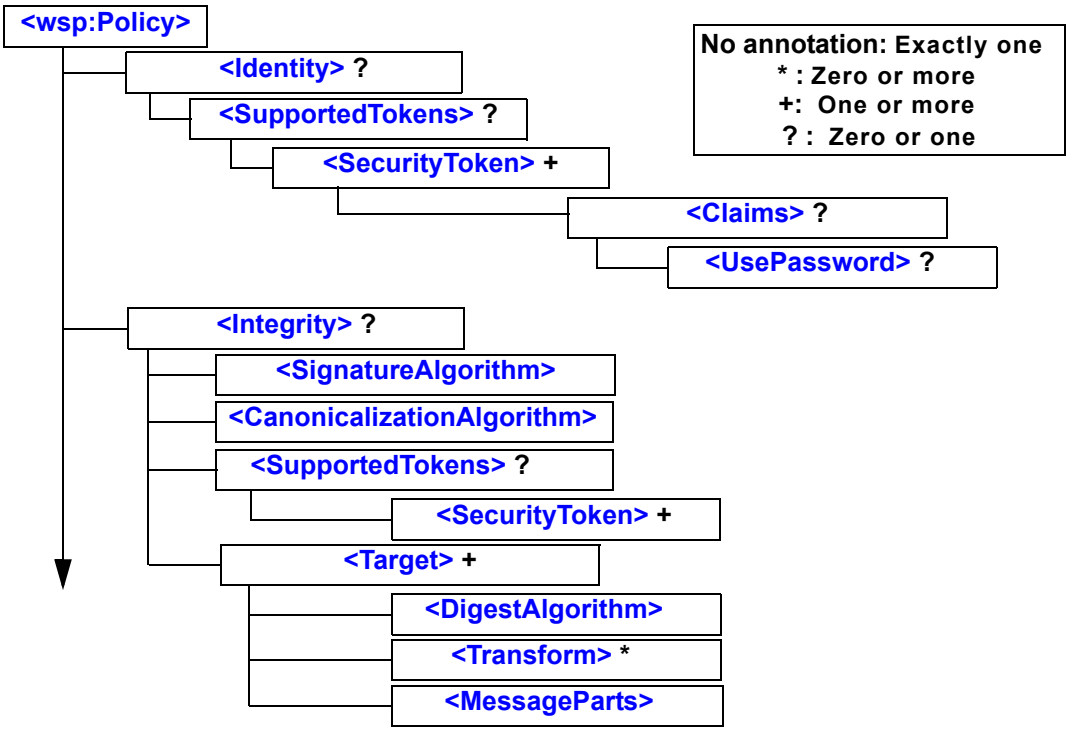
- Identity Assertion

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 BEA 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

BEA 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`.

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

BEA 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 <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-c14n#`	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 <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 <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 <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 <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 <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 <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 <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. BEA 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 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>
```