

Oracle® Service Architecture Leveraging Tuxedo (SALT)

Configuration Guide

11g Release 1 (11.1.1.2)

April 2011

ORACLE®

Copyright © 2006, 2011, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Oracle SALT Configuration Guide

Configuring Oracle Tuxedo Web Services	-1
Using Oracle Tuxedo Service Metadata Repository for Oracle SALT	-2
Defining Service-Level Keywords for Oracle SALT	-2
Defining Service Parameters for Oracle SALT	-6
Configuring Native Oracle Tuxedo Services	-8
Creating a Native WSDL	-9
Defining the SOAP Header	-9
Defining WSBinding Object	-11
Defining Service Object	-12
Configuring Message Conversion Handler	-12
Using WS-Policy Files	-13
Generating a WSDL File from a Native WSDL	-15
Configuring External Web Services	-15
Converting a WSDL file into Oracle Tuxedo Definitions	-16
WSDL-to-Tuxedo Service Metadata Keyword Mapping	-18
WSDL-to-WSDL Mapping	-18
Post Conversion Tasks	-19
Resolving Naming Conflict For the Generated Oracle SALT Proxy Service Definitions	-20
Loading the Generated SALT Proxy Service Metadata Definitions	-20
Setting Environment Variables for GWWS Runtime	-21
Creating the Oracle SALT Deployment File	-21
Importing the WSDL Files	-21
Configuring the GWWS Servers	-22
Configuring GWWS Server Level Properties	-23
Configuring Multiple Encoding Support	-25

Configuring System Level Resources	-27
Configuring Certificates	-27
Configuring Plug-in Libraries	-28
Configuring Advanced Web Service Messaging Features	-29
Web Service Addressing.	-29
Configuring the Addressing Endpoint for Outbound Services	-29
Disabling WS-Addressing.	-31
Web Service Reliable Messaging	-31
Creating the Reliable Messaging Policy File	-31
Specifying the Reliable Messaging Policy File in the WSDL File	-32
Configuring Security Features.	-33
Configuring Transport Level Security	-33
Setting Up SSL Link-Level Security	-33
Configuring Inbound HTTP Basic Authentication	-34
Configuring Outbound HTTP Basic Authentication.	-34
Configuring Message Level Web Service Security	-36
Main Use Cases of Web Service Security.	-36
Using WS-Security Policy Files	-36
Compiling SALT Configuration	-38
Configuring the UBBCONFIG File for Oracle SALT	-38
Configuring the TMMETADATA Server in the *SERVERS Section	-39
Configuring the GWWS Servers in the *SERVERS Section.	-40
Updating System Limitations in the UBBCONFIG File	-41
Configuring Certificate Password Phrase For the GWWS Servers	-42
Configuring Oracle Tuxedo Authentication for Web Service Clients	-43
Configuring Oracle Tuxedo Security Level for Outbound HTTP Basic Authentication.	-43
Configuring Oracle SALT In Oracle Tuxedo MP Mode	-44

Migrating from Oracle SALT 1.1	-44
Running GWWS servers with SALT 1.1 Configuration File.....	-44
Adopting SALT 2.0 Configuration Style by Converting SALT 1.1 Configuration File.....	-45
Configuring Oracle Tuxedo SCA Components	-47
Configuring an SCA ATMI Client.....	-48
Configuring an SCA JATMI Client.....	-49
Configuring an SCA Workstation Client.....	-50
Configuring an SCA Web Service Client	-51
Configuring an SCA ATMI Server	-54
Configuring an SCA Web Service Server	-56
Configuring SCA Client Security	-59
Oracle Tuxedo Application Domain Security	-59
Oracle Tuxedo Link-Level Security.....	-61
Configuring Link-Level Encryption	-61
Configuring Transport Layer Security	-62
Configuring Service Contract Discovery	-63
tpforward Support	-64
Service Contract Text File Output.....	-64
Examples	-66
Configuring Oracle SALT WS-TX Support.....	-67
Configuring Transaction Log Device	-68
Registration Protocol.....	-69
Configuring WS-TX Transactions.....	-69
Configuring Incoming Transactions.....	-70
Error Conditions	-70
Configuring Outbound Transactions	-70
Error Conditions	-71

Configuring Maximum Number of Transactions	-71
Configuring Policy Assertions	-72
Policy.xml File	-72
Inbound Transactions	-73
Outbound Transactions	-73
WSDL Generation	-73
WSDL Conversion	-73
See Also	-74

Oracle SALT Configuration Guide

This section contains the following topics:

- [Configuring Oracle Tuxedo Web Services](#)
- [Configuring Oracle Tuxedo SCA Components](#)
- [Configuring Service Contract Discovery](#)
- [Configuring Oracle SALT WS-TX Support](#)

Configuring Oracle Tuxedo Web Services

- [Using Oracle Tuxedo Service Metadata Repository for Oracle SALT](#)
- [Configuring Native Oracle Tuxedo Services](#)
- [Configuring External Web Services](#)
- [Configuring Service Contract Discovery](#)
- [Creating the Oracle SALT Deployment File](#)
- [Configuring Advanced Web Service Messaging Features](#)
- [Configuring Security Features](#)
- [Compiling SALT Configuration](#)
- [Configuring the UBBCONFIG File for Oracle SALT](#)

- [Configuring Oracle SALT In Oracle Tuxedo MP Mode](#)
- [Migrating from Oracle SALT 1.1](#)

Using Oracle Tuxedo Service Metadata Repository for Oracle SALT

Oracle SALT leverages the [Oracle Tuxedo Service Metadata Repository](#) to define service contract information for both Oracle Tuxedo legacy services and Oracle SALT proxy services. Service contract information for all listed Oracle Tuxedo services is obtained by accessing the Oracle Tuxedo Service Metadata Repository system service provided by the local Oracle Tuxedo domain. Typically, SALT calls the [TMMETADATA](#) system as follows:

- During GWSS server run-time.
 - It calls the Oracle Tuxedo Service Metadata Repository to retrieve necessary Oracle Tuxedo service definition at the appropriate time.
- When [tmwsdlgen](#) generates a WSDL file.
 - It calls the Oracle Tuxedo Service Metadata Repository to retrieve necessary Oracle Tuxedo service definitions and converts them to the WSDL description.

The following topics provide SALT-specific usage of Oracle Tuxedo Service Metadata Repository keywords and parameters:

- [Defining Service-Level Keywords for Oracle SALT](#)
- [Defining Service Parameters for Oracle SALT](#)

Defining Service-Level Keywords for Oracle SALT

[Table 1](#) lists the Oracle Tuxedo Service Metadata Repository service-level keywords used and interpreted by SALT.

Note: Metadata Repository service-level keywords that are not listed have no relevance to Oracle SALT and are ignored when SALT components load the Oracle Tuxedo Service Metadata Repository.

Table 1 Oracle SALT Usage of Service-Level Keywords in Oracle Tuxedo Service Metadata Repository

Service-Level Keyword	Oracle SALT Usage
service	<p>The unique key value of the service. This value is referenced in the SALT WSDL file.</p> <p>For native Oracle Tuxedo services, this value can be the same as the Oracle Tuxedo advertised service name or an alias name different from the actual Oracle Tuxedo advertised service name.</p> <p>For Oracle SALT proxy services, this value typically is the Web service operation local name.</p>
servicemode	<p>Determines the service mode (i.e., native Oracle Tuxedo service or Oracle SALT proxy service).</p> <p>The valid values are:</p> <ul style="list-style-type: none"> • <code>tuxedo</code> represents a native Oracle Tuxedo service • <code>webservice</code> represents an Oracle SALT proxy service, i.e. a service definition converted from a <code>wsdl:operation</code> <p>Do not use “webservice” to define a native Oracle Tuxedo service. This value is always used to define services converted from external Web services.</p>
tuxservice	<p>The actual Oracle Tuxedo advertised service name. If no value is specified, then the value is the same as the value in the <code>service</code> keyword.</p> <p>For native Oracle Tuxedo service, Oracle SALT invokes the Oracle Tuxedo service defined using this keyword.</p> <p>For Oracle SALT proxy service, GWWS server advertises the service name using this keyword value.</p>
servicetype	<p>Determines the service message exchange pattern for the specified Oracle Tuxedo service.</p> <p>The following values specify mapping rules between the Oracle Tuxedo service types and Web Service message exchange pattern (MEP):</p> <ul style="list-style-type: none"> • <code>service</code> corresponds to request-response MEP • <code>oneway</code> corresponds to oneway request MEP • <code>queue</code> corresponds to request-response MEP

Table 1 Oracle SALT Usage of Service-Level Keywords in Oracle Tuxedo Service Metadata Repository

Service-Level Keyword	Oracle SALT Usage
inbuf	<p>Specifies the input buffer (request buffer) type for the service.</p> <p>For native Oracle Tuxedo services, the value can be any Oracle Tuxedo typed buffer type. The following values are Oracle Tuxedo reserved buffer types:</p> <p>STRING, CARRAY, XML, MBSTRING, VIEW, VIEW32, FML, FML32, X_C_TYPE, X_COMMON, X_OCTET, NULL (input buffer is empty)</p> <p>Note: The value is case sensitive, if inbuf specifies any other type other than the previous buffer types, the buffer is treated as a custom buffer type.</p> <p>For Oracle SALT proxy services, the value is always FML32.</p>
outbuf	<p>Specifies the output buffer (response buffer with TPSUCCESS) type for the service.</p> <p>For native Oracle Tuxedo services, the value can be any Oracle Tuxedo typed buffer type. The following values are Oracle Tuxedo reserved buffer types:</p> <p>STRING, CARRAY, XML, MBSTRING, VIEW, VIEW32, FML, FML32, X_C_TYPE, X_COMMON, X_OCTET, NULL (input buffer is empty)</p> <p>Note: The value is case sensitive, if outbuf specifies any other type other than the previous buffer types, the buffer is treated as a custom buffer type.</p> <p>For Oracle SALT proxy services, the value is always FML32.</p>

Table 1 Oracle SALT Usage of Service-Level Keywords in Oracle Tuxedo Service Metadata Repository

Service-Level Keyword	Oracle SALT Usage
errbuf	<p>Specifies the error buffer (response buffer with TPFALL) type for the service.</p> <p>For native Oracle Tuxedo services, the value can be any Oracle Tuxedo typed buffer type. The following values are Oracle Tuxedo reserved buffer types:</p> <p>STRING, CARRAY, XML, MBSTRING, VIEW, VIEW32, FML, FML32, X_C_TYPE, X_COMMON, X_OCTET, NULL (input buffer is empty)</p> <p>Note: The value is case sensitive, if <code>errbuf</code> specifies any other type other than the previous buffer types, the buffer is treated as a custom buffer type.</p> <p>For Oracle SALT proxy services, the value is always <code>FML32</code>.</p>
inview	<p>Specifies the view name used by the service for the following input buffer types:</p> <p>VIEW, VIEW32, X_C_TYPE, X_COMMON</p> <p>Oracle SALT requires that you specify the view name rather than accept the default <code>inview</code> setting.</p> <p>This keyword is for native Tuxedo services only.</p>
outview	<p>Specifies the view name used by the service for the following output buffer types:</p> <p>VIEW, VIEW32, X_C_TYPE, X_COMMON</p> <p>Oracle SALT requires that you specify the view name rather than accept the default <code>outview</code> setting.</p> <p>This keyword is for native Oracle Tuxedo services only.</p>
errview	<p>Specifies the view name used by the service for the following error buffer types:</p> <p>VIEW, VIEW32, X_C_TYPE, X_COMMON</p> <p>Oracle SALT requires that you specify the view name rather than accept the default <code>errview</code> setting.</p> <p>This keyword is for native Oracle Tuxedo services only.</p>

Table 1 Oracle SALT Usage of Service-Level Keywords in Oracle Tuxedo Service Metadata Repository

Service-Level Keyword	Oracle SALT Usage
<code>inbufschema</code>	<p>Specifies external XML Schema element associated with the service input buffer. If this value is specified, Oracle SALT incorporates the external schema in the generated WSDL to replace the default data type mapping rule for the service input buffer.</p> <p>This keyword is for native Oracle Tuxedo services only.</p>
<code>outbufschema</code>	<p>Specifies external XML Schema element associated with the service output buffer. If this value is specified, Oracle SALT incorporates the external schema in the generated WSDL to replace the default data type mapping rule for the service output buffer.</p> <p>This keyword is for native Oracle Tuxedo services only.</p>
<code>errbufschema</code>	<p>Specifies external XML Schema element associated with the service error buffer. If this value is specified, Oracle SALT incorporates the external schema in the generated WSDL to replace the default data type mapping rule for the service error buffer.</p> <p>This keyword is for native Oracle Tuxedo services only.</p>

Defining Service Parameters for Oracle SALT

The Oracle Tuxedo Service Metadata Repository interprets parameters as sub-elements encapsulated in an Oracle Tuxedo service typed buffer. Each parameter can have its own data type, occurrences in the buffer, size restrictions, and other Oracle Tuxedo-specific restrictions. Please note:

- **VIEW, VIEW32, X_C_TYPE, or X_COMMON** typed buffers
Each parameter of the buffer should represent a VIEW/VIEW32 structure member.
- **FML or FML32** typed buffers
Each parameter of the buffer should represent an FML/FML32 field element that may be present in the buffer.
- **STRING, CARRAY, XML, MBSTRING, and X_OCTET** typed buffers
Oracle Tuxedo treats these buffers holistically. At most, one parameter is permitted for the buffer to define restriction facets (such as buffer size threshold).
- **Custom** typed buffers

Parameters facilitate describing details about the buffer type.

- FML32 typed buffers that support embedded VIEW32 and FML32 buffers

Embedded parameters provide support.

Table 2 lists the Oracle Tuxedo Service Metadata Repository parameter-level keywords used and interpreted by SALT.

Note: Metadata Repository parameter-level keywords that are not listed have no relevance to Oracle SALT and are ignored when SALT components load the Oracle Tuxedo Service Metadata Repository.

Table 2 Oracle SALT Usage of Parameter-Level Keyword in Oracle Tuxedo Service Metadata Repository

Parameter-level Keyword	Oracle SALT Usage
param	<p>Specifies the parameter name.</p> <ul style="list-style-type: none"> • VIEW, VIEW32, X_C_TYPE, or X_COMMON <p>Specifies the view structure member name in the param keyword.</p> <ul style="list-style-type: none"> • FML, FML32 <p>Specifies the FML/FML32 field name in the param keyword.</p> <ul style="list-style-type: none"> • STRING, CARRAY, XML, MBSTRING, or X_OCTET <p>Oracle SALT ignores the parameter definitions.</p>
type	<p>Specifies the data type of the parameter.</p> <p>Note: Oracle SALT does not support <code>dec_t</code> and <code>ptr</code> data types.</p>
subtype	<p>Specifies the view structure name if the parameter type is <code>view32</code>. For any other typed parameter, Oracle SALT ignores this value.</p> <p>Note: Oracle SALT requires this value if the parameter type is <code>view32</code>.</p> <p>This keyword is for native Oracle Tuxedo service only.</p>
access	<p>The general definition applies for this parameter. To support Oracle Tuxedo TPFAIL scenario, the <code>access</code> attribute value has been enhanced.</p> <p>Original values: <code>in</code>, <code>out</code>, <code>inout</code>, <code>noaccess</code>.</p> <p>New added values: <code>err</code>, <code>inerr</code>, <code>outerr</code>, <code>inouterr</code>.</p>

Table 2 Oracle SALT Usage of Parameter-Level Keyword in Oracle Tuxedo Service Metadata Repository

Parameter-level Keyword	Oracle SALT Usage
count	The general definition applies for this parameter. For Oracle SALT, the value for the <code>count</code> parameter must be greater than or equal to <code>requiredcount</code> .
requiredcount	The general definition applies for this parameter. The default is 1. For Oracle SALT, the value for the <code>count</code> parameter must be greater than or equal to <code>requiredcount</code> .
size	This optional keyword restricts the maximum byte length of the parameter. It is only valid for the following parameter types: <code>STRING</code> , <code>CARRAY</code> , <code>XML</code> , and <code>MBSTRING</code> If this keyword is not set, there is no maximum byte length restriction for this parameter. The value range is [0 , 2147483647]
paramschema	Specifies the corresponding XML Schema element name of the parameter. It is generated by the Oracle SALT WSDL converter. This keyword is for Oracle SALT proxy service only. Do not specify this keyword for native Oracle Tuxedo services.
primetype	Specifies the corresponding XML primitive data type of the parameter. It is generated by Oracle SALT WSDL converter according to Oracle SALT pre-defined XML-to-Tuxedo data type mapping rules. This keyword is for Oracle SALT proxy service only. Do not specify this keyword for native Oracle Tuxedo services.

Configuring Native Oracle Tuxedo Services

This section describes the required and optional configuration tasks for exposing native Oracle Tuxedo services as Web services:

- [Creating a Native WSDL](#)
- [Using WS-Policy Files](#)
- [Generating a WSDL File from a Native WSDL](#)

Creating a Native WSDL

To expose a set of Oracle Tuxedo services as Web services through one or more HTTP/S endpoints, a native WSDL must be defined.

Each native WSDL must be defined with a unique WSDL name. A WSDL can define one or more <WSBinding> elements for more Web service application details (such as SOAP protocol details, the Oracle Tuxedo service list to be exposed as web service operations, and so on).

This section contains the following topics:

- [Defining the SOAP Header](#)
- [Defining WSBinding Object](#)
- [Defining Service Object](#)
- [Configuring Message Conversion Handler](#)

Defining the SOAP Header

The `mapsoapheader` attribute is used to configure SOAP headers. It defines an FML32 field that represents the SOAP header. It is `TA_WS_SOAP_HEADER STRING` type.

Note: The `mapsoapheader` attribute is defined in `wsoapflds.h` file shipped with Oracle SALT.

[Listing 1](#) shows a SOAP header definition example.

Listing 1 SOAP Header Definition

```
<Definition ...>
  <WSBinding id="simpapp_binding">
    <Servicegroup id="simpapp">
      <Service name="toupper">
        <Property name="mapsoapheader" value="true" />
      </Service>
    </Servicegroup>
    ....
  </WSBinding>
```

```
</Definition>
```

The `mapsoapheader` attribute default value is "false" which indicates the GWWS does not execute mapping between the SOAP header and FML fields.

If `mapsoapheader` is set to `true` the mapping behavior is as follows for inbound and outbound services:

- Inbound

For inbound services, the GWWS translates the SOAP header as shown in [GWWS Soap Header Translation](#)[GWWS Soap Header Translation](#)[GWWS Soap Header Translation](#)[Listing 2](#).

Listing 2 GWWS Soap Header Translation

```
<cup:SoapHeader
xmlns:cup='http://www.chinaunionpay.com/soa/esb/message/1_0'>
<cup:Head>
    <cup:Name>xxx</cup:Name>
    <cup:Value>xxx</cup:Value>
</cup:Head>
</cup:SoapHeader>
```

The string buffer is assigned to the `TA_WS_SOAP_HEADER` field and injects the target FML32 buffer. If the target buffer type is not FML32, the translation will not take effect.

- Out Bound

For outbound services, the GWWS receives the `TA_WS_SOAP_HEADER` from the request buffer and places it in the SOAP header when the SOAP package is composed.

Defining WSBinding Object

Each WSBinding object is defined using the <WSBinding> element. Each WSBinding object must be defined with a unique WSBinding id within the WSDL. The WSBinding id is a required indicator for the SALTDEPLOY file reference used by the GWWS.

Each WSBinding object can be associated with SOAP protocol details by using the <SOAP> sub-element. By default, SOAP 1.1, *document/literal* styled SOAP messages are applied to the WSBinding object.

[Listing 3](#) shows how SOAP protocol details are redefined using the <SOAP> sub-element.

Listing 3 Defining SOAP Protocol Details for a WSBinding

```
<Definition ...>
  <WSBinding id="simpapp_binding">
    <Servicegroup id="simpapp">
      <Service name="toupper" />
      <Service name="tolower" />
    </Servicegroup>
    <SOAP version="1.2" style="rpc" use="encoded">
      <AccessingPoints>
        ...
      </AccessingPoints>
    </SOAP>
  </WSBinding>
</Definition>
```

Within the <SOAP> element, a set of access endpoints can be specified. The URL value of these access endpoints are used by corresponding GWWS servers to create the listen HTTP/S protocol port. It is recommended to specify one HTTP and HTTPS endpoint (at most) for each GWWS server for an *inbound* WSBinding object.

Each WSBinding object must be defined with a group of Oracle Tuxedo services using the <Servicegroup> sub-element. Each <Service> element under <Servicegroup> represents an Oracle Tuxedo service that can be accessed from a Web service client.

Defining Service Object

Each service object is defined using the `<Service>` element. Each service must be specified with the “name” attribute to indicate which Oracle Tuxedo service is exposed. Usually, the “name” value is used as the key value for obtaining Oracle Tuxedo service contract information from the Oracle Tuxedo Service Metadata Repository.

[Listing 4](#) shows how a group of services are defined for `WSBinding`.

Listing 4 Defining a Group of Services for a `WSBinding`

```
<Definition ...>
  <WSBinding id="simpapp_binding">
    <Servicegroup id="simpapp">
      <Service name="toupper" />
      <Service name="tolower" />
    </Servicegroup>
    ...
  </WSBinding>
</Definition>
```

Configuring Message Conversion Handler

You can create your own plug-in functions to customize SOAP XML payload and Oracle Tuxedo typed buffer conversion routine. For more information, see [Using Oracle SALT Plug-ins in Oracle SALT Programming Web Services](#) and [“Configuring Plug-in Libraries” on page 1-28](#).

Once a plug-in is created and configured, it can be referenced using the `<service>` element to specify user-defined data mapping rules for that service. The `<Msghandler>` element can be defined at the message level (`<Input>`, `<Output>` or `<Fault>`) to specify which implementation of “P_CUSTOM_TYPE” category plug-in should be used to do the message conversion. The `<Msghandler>` element content is the Plug-in name.

[Listing 5](#) shows a service that uses the “MBCONV” custom plug-in to convert input and “XMLCONV” custom plug-in to convert output.

Listing 5 Configuring Message Conversion Handler for a Service

```

<Definition ...>
  <WSBinding id="simpapp_binding">
    <Servicegroup id="simpapp">
      <Service name="toupper" >
        <Input>
          <Msghandler>MBCONV</Msghandler>
        </Input>
        <Output>
          <Msghandler>XMLCONV</Msghandler>
        </Output>
      </Service>
    </Servicegroup>
    ...
  </WSBinding>
</Definition>

```

Using WS-Policy Files

Advanced Web service features can be enabled by configuring WS-Policy files (for example, Reliable Messaging and Web Service Message-Level Security). You may need to create WS-Policy files to use these features. The [Web Service Policy Framework specifications](#) provides a general purpose model and syntax to describe and communicate the policies of a Web Service.

To use WS-Policy files, the `<Policy>` element should be defined in the WSDL to incorporate these separate WS-Policy files. Attribute `location` is used to specify the policy file path, both abstract and relative file path are allowed. Attribute `use` is optionally used by message level assertion policy files to specify the applied messages, request (input) message, response (output) message, fault message, or the combination of the three.

There are two different sub-elements in the WSDL that reference WS-Policy files:

- `<Servicegroup>`
 - If a WS-Policy file consists of Web Service Endpoint level Assertions, e.g. Reliable Messaging Assertion, the WS-Policy file applies to all endpoints that serving this `<Servicegroup>`.

- If a WS-Policy file consists of Web Service Operation level Assertions, e.g., Security Identity Assertion, the WS-Policy file applies to all services listed in this <Servicegroup>.
- If a WS-Policy file consists of Web Service Message level Assertions, e.g., Security SignedParts Assertion, the WS-Policy file applies to input, output and/or fault messages of all services listed in this <Servicegroup>.
 - Note: Oracle SALT only supports request message level assertions for the current release. You must only specify `use="input"` for message level assertion policy files.
- <Service>
 - If a WS-Policy file consists of Web Service Operation level Assertions, e.g. Security Identity Assertion, the WS-Policy file applies to this particular service.
 - If a WS-Policy file consists of Web Service Message level Assertions, e.g. Security SignedParts Assertion, the WS-Policy file applies to input, output and/or fault messages of this particular service.
 - Note: Oracle SALT only supports request message level assertions for the current release. You must only specify `use="input"` for message level assertion policy files.

Oracle SALT provides some pre-packaged WS-Policy files for most frequently used cases. These WS-Policy files are located under directory `$TUXDIR/udataobj/salt/policy`. These files can be referenced using `location="salt:<policy_file_name>"`.

[Listing 6](#) shows a sample of using WS-Policy Files in the native WSDL file.

Listing 6 A Sample of Defining WS-Policy Files in the WSDL File

```
<Definition ...>
  <WSBinding id="simpapp_binding">
    <Servicegroup id="simpapp">
      <Policy location="./endpoint_policy.xml" />
      <Policy location="/usr/resc/all_input_msg_policy.xml" use="input" />
      <Service name="toupper">
        <Policy location="service_policy.xml" />
        <Policy location="/usr/resc/input_message_policy.xml"
          use="input" />
      </Service>
    </Servicegroup>
  </WSBinding>
</Definition>
```

```

        <Service name="tolower" />
    </Servicegroup>
    ....
</WSBinding>
</Definition>

```

For more information, see [“Specifying the Reliable Messaging Policy File in the WSDL File”](#) and [“Using WS-Security Policy Files”](#).

Generating a WSDL File from a Native WSDL

Once an Oracle Tuxedo native WSDL is created, the corresponding WSDL file can be generated using the Oracle SALT WSDL generation utility, `tmwsdlgen`. The following example command generates a WSDL file named “`app1.wsdl`” from a given WSDL named “`app1.wsdf`”:

```
tmwsdlgen -c app1.wsdf -o app1.wsdl
```

Note: Before executing `tmwsdlgen`, the `TUXCONFIG` environment variable must be set correctly and the relevant Oracle Tuxedo application using `TMMETADATA` must be booted.

You can optionally specify the output WSDL file name using the ‘`-o`’ option. Otherwise, `tmwsdlgen` creates a default WSDL file named “`tuxedo.wsdl`”.

If the native WSDL file contains Oracle Tuxedo services that use `CARRAY` buffers, you can specify `tmwsdlgen` options to generate different styled WSDL files for `CARRAY` buffer mapping. By default, `CARRAY` buffers are mapped as `xsd:base64Binary` XML data types in the SOAP message. For more information, see [Data Type Mapping and Conversions](#) in *Oracle SALT Programming Web Services* and [tmwsdlgen](#) in the *Oracle SALT Reference Guide*.

Configuring External Web Services

To invoke an external Web Service from Oracle Tuxedo, the following configuration tasks need to be performed:

- [Converting a WSDL file into Oracle Tuxedo Definitions](#)
- [Post Conversion Tasks](#)

Converting a WSDL file into Oracle Tuxedo Definitions

Oracle SALT provides a WSDL conversion command utility to convert external WSDL files into Oracle Tuxedo definitions. The WSDL file is converted using Extensible Stylesheet Language Transformations (XSLT) technology. Apache Xalan Java 2.7.0 is bundled in the Oracle SALT installation package and is used as the default XSLT toolkit.

Oracle SALT WSDL converter is composed of two parts:

- The xsl files, which process the WSDL file.
- The command utility, `wsdlcvt`, invokes the Xalan toolkit. This wrapper script provides a user friendly WSDL Converter interface.

The following sample command converts an external WSDL file and generates Oracle Tuxedo definition files.

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

[Table 3](#) lists the Oracle Tuxedo definition files generated by Oracle SALT WSDL Converter.

Table 3 Tuxedo Definition Files generated by Oracle SALT WSDL Converter

Generated File	Description
Oracle Tuxedo Service Metadata Repository input file	<p>Oracle SALT WSDL Converter converts each <code>wsdl:operation</code> to a Oracle Tuxedo service metadata syntax compliant service called Oracle SALT proxy service. Oracle SALT proxy services are advertised by GWWS servers to accept ATMI call from Oracle Tuxedo applications.</p>
FML32 field table definition file	<p>Oracle SALT maps each <code>wsdl:message</code> to an Oracle Tuxedo FML32 typed buffer. Oracle SALT WSDL Converter decomposes XML Schema of each message and maps each basic XML snippet as an FML32 field. The generated FML32 fields are defined in a definition table file, and the field name equals to the XML element local name by default.</p> <p>To access an Oracle SALT proxy service, Oracle Tuxedo applications must refer to the generated FML32 fields to handle the request and response message. FML32 environment variables must be set accordingly so that both Oracle Tuxedo applications and GWWS servers can map between field names and field identifier values.</p> <p>Note: You may want to re-define the generated field names due to field name conflict or some other reason. In that case, both Oracle Tuxedo Service Metadata Definition input file and FML32 field table definition file must be changed accordingly. For more information, see “Resolving Naming Conflict For the Generated Oracle SALT Proxy Service Definitions”.</p>
Non-native WSDF file	<p>Oracle SALT WSDL Converter converts the WSDL file into a WSDF file, which can be deployed to GWWS servers in the Oracle SALT deployment file for outbound direction. The generated WSDF file is so-called non-native WSDF file.</p> <p>Note: Please do not deploy non-native WSDF files for inbound direction.</p>
XML Schema files	<p>WSDL embedded XML Schema and imported XML Schema (XML Schema content referenced with <code><xsd:import></code>) are saved locally as <code>.xsd</code> files. These files are used by GWWS servers and need to be saved under the same directory.</p> <p>Note: New XML Schema environment variables <code>XSDDIR</code> and <code>XSDFILES</code> must be set accordingly so that GWWS servers can load these <code>.xsd</code> files.</p>

WSDL-to-Tuxedo Service Metadata Keyword Mapping

Table 4 lists WSDL Element-to-Tuxedo Service Metadata Definition Keyword mapping rules.

Table 4 WSDL Element-to-Tuxedo Service Metadata Definition Mapping

WSDL Element	Corresponding Oracle Tuxedo Service Metadata Definition Keyword	Note
/wsdl:definitions /wsdl:portType /wsdl:operation @name	service	Oracle SALT proxy service name. The keyword value equals to the operation local name.
	tuxservice	Oracle SALT proxy service advertised name in Oracle Tuxedo system. If the wsdl operation local name is less than 15 characters, keyword value equals to the operation local name, otherwise the keyword value is the first 15 characters of the operation local name.
/wsdl:definitions /wsdl:portType /wsdl:operation /wsdl:input	inbuf=FML32	WSDL operation messages are always mapped as Oracle Tuxedo FML32 buffer types. Please do not change the buffer type any way.
/wsdl:definitions /wsdl:portType /wsdl:operation /wsdl:output	outbuf=FML32	Note: For more information about wsdl message and FML32 buffer mapping, see XML-to-Tuxedo Data Type Mapping for External Web Services in the <i>Oracle SALT Programming Web Services</i> .
/wsdl:definitions /wsdl:portType /wsdl:operation /wsdl:fault	errbuf=FML32	

WSDL-to-WSDF Mapping

Table 5 lists WSDL Element-to-WSDF Element mapping rules.

Table 5 WSDL Element-to-WSDL Element Mapping

WSDL Element	WSDL Element	Note
/wsdl:definitions @targetNamespace	/Definition @wsdlNamespace	Each wsdl:definition maps to a WSDL Definition.
/wsdl:definitions /wsdl:binding	/Definition /WSBinding	Each wsdl:binding object maps to a WSDL WSBinding element.
/wsdl:definitions /wsdl:binding @type	/Definition /WSBinding /Servicegroup	Each wsdl:binding referenced wsdl:portType object maps to the Servicegroup element of the corresponding WSBinding element.
/wsdl:definitions /wsdl:binding /soap:binding	/Definition /WSBinding /SOAP @version	If namespace prefix “soap” refers to URI “http://schemas.xmlsoap.org/wsdl/soap/”, the SOAP version attribute value is “1.1”; If namespace prefix “soap” refers to URI “http://schemas.xmlsoap.org/wsdl/soap12/”, the SOAP version attribute value is “1.2”.
/wsdl:definitions /wsdl:binding /soap:binding @style	/Definition /WSBinding /SOAP @style	The WSDL WSBinding SOAP message style setting equals to the corresponding WSDL soap binding message style setting (“rpc” or “document”).
/wsdl:definitions /wsdl:binding /wsdl:operation	/Definition /WSBinding /Servicegroup /Service	Each wsdl:operation object maps to a Service element of the corresponding WSBinding element.
/wsdl:definitions /wsdl:port /soap:address	/Definition /WSBinding /SOAP /AccessingPoints /Endpoint	Each soap:address endpoint defined for a wsdl:binding object maps to a Endpoint element of the corresponding WSBinding element.

Post Conversion Tasks

The following post conversion tasks need to be performed for configuring outbound Web service applications:

- [Resolving Naming Conflict For the Generated Oracle SALT Proxy Service Definitions](#)
- [Loading the Generated SALT Proxy Service Metadata Definitions](#)
- [Setting Environment Variables for GWWS Runtime](#)

Resolving Naming Conflict For the Generated Oracle SALT Proxy Service Definitions

When converting a WSDL file, unexpected naming conflicts may be found due to truncation or lost context information. Before using the generated Service Metadata Definitions and FML32 field table files, the following potential naming conflicts must be eliminated first.

- Eliminating the duplicated service metadata keyword “`tuxservice`” definitions

The keyword `tuxservice` in the Oracle SALT proxy service metadata definition is the truncated value of the original Web Service operation local name if the operation name is more than 15 characters. The truncated `tuxservice` value may be duplicated for multiple Oracle SALT proxy service entries. Since GWWS server uses `tuxservice` values as the advertised service names, so you must manually resolve the naming conflict among multiple Oracle SALT proxy services to avoid uncertain service request delivery. To resolve the naming conflict, you should assign a unique and meaningful name to `tuxservice`.

- Eliminating the duplicated FML32 field definitions

When converting a external WSDL file into Oracle Tuxedo definitions, each `wsdl:message` is parsed and mapped as an FML32 buffer format which containing a set of FML32 fields to represent the basic XML snippets of the `wsdl:message`. By default, The generated FML32 fields are named using the corresponding XML element local names.

The FML32 field definitions in the generated field table file are sorted by field name so that duplicated names can be found easily. In order to achieve a certain SOAP/FML32 mapping, the field name conflicts must be resolved. You should modify the generated duplicated field name with other unique and meaningful FML32 field name values. The corresponding Service Metadata Keyword `param` values in the generated Oracle SALT proxy service definition must be modified accordingly. The generated comments of the FML32 fields and Service Metadata Keyword “`param`” definitions are helpful in locating the corresponding `name` and `param`.

Loading the Generated SALT Proxy Service Metadata Definitions

After potential naming conflicts are resolved, you should load the Oracle SALT proxy service metadata definitions into the Oracle Tuxedo Service Metadata Repository through `tmlloadrepos`

utility. For more information, see `tmloadrepos`, in the [Oracle Tuxedo Service Metadata Repository Documentation](#).

Setting Environment Variables for GWWS Runtime

Before booting GWWS servers for outbound Web services, the following environment variable settings must be performed.

- Update environment variable `FLDTBLDIR32` and `FIELDTBLS32` to add the generated FML32 field table files.
- Place all excerpted XML Schema files into one directory, and set environment variable `XSDDIR` and `XSDFILES` accordingly.
 - Environment variable `XSDDIR` and `XSDFILES` are introduced in the SALT 2.0 release. They are used by the GWWS server to load all external XML Schema files at run time. Multiple XML Schema file names should be delimited with comma ‘,’. For instance, if you placed XML Schema files: `a.xsd`, `b.xsd` and `c.xsd` in directory `/home/user/myxsd`, you must set environment variable `XSDDIR` and `XSDFILES` as follows before booting the GWWS server:

```
XSDDIR=/home/user/myxsd
XSDFILES=a.xsd,b.xsd,c.xsd
```

Creating the Oracle SALT Deployment File

The Oracle SALT Deployment file (`SALTDEPLOY`) defines a SALT Web service application. The `SALTDEPLOY` file is the major input for Web service application in the binary `SALTCONFIG` file.

To create a `SALTDEPLOY` file, do the following steps:

1. [Importing the WSDF Files](#)
2. [Configuring the GWWS Servers](#)
3. [Configuring System Level Resources](#)

For more information, see [Oracle SALT Deployment File Reference](#) in the Oracle SALT Reference Guide.

Importing the WSDF Files

You should import all your required WSDF files to the Oracle SALT deployment file. Each imported WSDF file must have a unique WSDF name which is used by the GWWS servers to make

deployment associations. Each imported WSDL file must be accessible through the location specified in the SALTDEPLOY file.

[Listing 7](#) shows how to import WSDL files in the SALTDEPLOY file.

Listing 7 Importing WSDL Files in the SALTDEPLOY File

```
<Deployment ..>
  <WSDL>
    <Import location="/home/user/simpapp_wsdl.xml" />
    <Import location="/home/user/rmapp_wsdl.xml" />
    <Import location="/home/user/google_search.wsdl" />
  </WSDL>
  ...
</Deployment>
```

Configuring the GWWS Servers

Each GWWS server can be deployed with a group of inbound WSBinding objects and a group of outbound WSBinding objects defined in the imported WSDL files. Each WSBinding object is referenced using attribute “ref=<wsdl_name>:<WSBinding id>”. For inbound WSBinding objects, each GWWS server must specify at least one access endpoint as an inbound endpoint from the endpoint list in the WSBinding object. For outbound WSBinding objects, each GWWS server can specify zero or more access endpoints as outbound endpoints from the endpoint list in the WSBinding object.

[Listing 8](#) shows how to configure GWWS servers with both inbound and outbound endpoints.

Listing 8 GWWS Server Defined In the SALTDEPLOY File

```
<Deployment ..>
  ...
  <WSGateway>
    <GWInstance id="GWWS1">
      <Inbound>
        <Binding ref="appl:appl_binding">
          <Endpoint use="simpapp_GWWS1_HTTPPort" />
        </Binding>
      </Inbound>
    </GWInstance>
  </WSGateway>
</Deployment>
```

```

        <Endpoint use="simpapp_GWWS1_HTTPSPort" />
    </Binding>
</Inbound>
<Outbound>
    <Binding ref="app2:app2_binding">
        <Endpoint use=" simpapp_GWWS1_HTTPPort" />
        <Endpoint use=" simpapp_GWWS1_HTTPSPort" />
    </Binding>
    <Binding ref="app3:app3_binding" />
</Outbound>
</GWInstance>
</WSGateway>
...
</ Deployment>

```

Configuring GWWS Server Level Properties

The GWWS server can be configured with properties that switch feature on/off or set argument to tune the server’s performance.

Properties are configured in the <GWInstance> child element <Properties>. Each individual property is defined by using the <Property> element which contains a “name” attribute and a “value” attribute). Different “name” attributes represent different property elements that contain a value. [Table 6](#) lists GWWS server level properties.

Table 6 GWWS Server Level Properties

Property Name	Description	Value Range	Default
enableMultiEncoding	Switch on/off the SOAP message multiple encoding support	"true" "false"	"false"
max_backlog	Specify socket backlog control value	[1, 255]	20
max_content_length	Specify the maximum allowed incoming HTTP message content length.	[0, 1G](byte) (Can set suffix 'M', 'G', e.g. 1.5M, 0.2G)	0 (means no limit)

Table 6 GWWS Server Level Properties

Property Name	Description	Value Range	Default
thread_pool_size	Specify the GWWS server thread pool size.	[1, 1024]	16
timeout	Specify the network timeout in seconds.	[1, 65535] (unit:sec)	300
wasm_acktime	Specify the Reliable Messaging Acknowledgement message reply policy. GWWS servers support replying acknowledgement messages either after receiving the SOAP request from network immediately or after the Oracle Tuxedo service returns the response message.	"NETRECV" "REPLYRECV"	"NETRECV"

Note: For more information about GWWS multiple encoding support, see [“Configuring Multiple Encoding Support”](#).

For more information about Performance tuning properties, see “Tuning the GWWS Server” in [Administering Oracle SALT at Runtime](#).

[Listing 9](#) shows an example of how GWWS properties are configured.

Listing 9 Configuring GWWS Server Properties

```
<Deployment ...>
...
<WSGateway>
  <GWInstance id="GWWS1">
    .....
    <Properties>
      <Property name="thread_pool_size" value="20"/>
      <Property name="enableMultiEncoding" value="true"/>
      <Property name="timeout" value="600"/>
    </Properties>
  </GWInstance>
</WSGateway>
```



```
...
</ Deployment>
```

Configuring Multiple Encoding Support

Oracle SALT supports multiple encoding SOAP messages and the encoding mappings between SOAP message and Oracle Tuxedo buffer. Oracle SALT supports the following character encoding:

```
ASCII, BIG5, CP1250, CP1251, CP1252, CP1253, CP1254, CP1255, CP1256,
CP1257, CP1258, CP850, CP862, CP866, CP874, EUC-CN, EUC-JP, EUC-KR,
GB18030, GB2312, GBK, ISO-2022-JP, ISO-8859-1, ISO-8859-13,
ISO-8859-15, ISO-8859-2, ISO-8859-3, ISO-8859-4, ISO-8859-5,
ISO-8859-6, ISO-8859-7, ISO-8859-8, ISO-8859-9, JOHAB, KOI8-R,
SHIFT_JIS, TIS-620, UTF-16, UTF-16BE, UTF-16LE, UTF-32, UTF-32BE,
UTF-32LE, UTF-7, UTF-8
```

To enable the GWWS multiple encoding support, GWWS server level property “enableMultiEncoding” should be set to “true” as shown in [Listing 10](#).

Note: GWWS internally converts non UTF-8 external messages into UTF-8. However, encoding conversion hurts server performance. By default, encoding conversion is turned off and messages that are not UTF-8 encoded are rejected.

Listing 10 Configuring GWWS Server Multiple Encoding Property

```
<Deployment ..>
...
<WSGateway>
  <GWInstance id="GWWS1">
    .....
    <Properties>
      <Property name="enableMultiEncoding" value="true"/>
    </Properties>
  </GWInstance>
</WSGateway>
...
</ Deployment>
```

[Table 7](#) explains the detailed SOAP message and Oracle Tuxedo buffer encoding mapping rules if the GWWS server level multiple encoding switch is turned on.

Table 7 Oracle SALT Message Encoding Mapping Rules

Mapping from ...	Mapping to ...	Encoding Mapping Rule
SOAP/XML	Oracle Tuxedo Typed Buffer	<code>string/mbstring/xml</code> buffer or field characters' encoding equals to SOAP xml encoding.
STRING Typed Buffer	SOAP/XML	GWWS sets the target SOAP message in UTF-8 encoding, and assumes the original <code>STRING</code> buffer containing only UTF-8 encoding characters. Note: Oracle Tuxedo Developers must ensure the <code>STRING</code> characters are in UTF-8 encoding.
MBSTRING/XML Typed Buffer	SOAP/XML	SOAP xml encoding equals to <code>MBSTRING/XML</code> encoding.
FML/32, VIEW/32 Typed Buffer that containing the same encoding setting for multiple <code>FLD_MBSTRING</code> fields	SOAP/XML	SOAP xml encoding is set to <code>FLD_MBSTRING</code> encoding, the original Typed buffer field characters are not changed in the SOAP message. Note: Oracle Tuxedo Developers must ensure the <code>FLD_STRING</code> characters in the same buffer are in consistent encoding.
FML/32, VIEW/32 Typed Buffer that containing the different encoding for multiple <code>FLD_MBSTRING</code> fields	SOAP/XML	SOAP xml encoding is set to UTF-8, the original Typed buffer <code>FLD_MBSTRING</code> field characters in other encoding are converted into UTF-8 in the SOAP message. Note: Oracle Tuxedo Developers must ensure the <code>FLD_STRING</code> characters in the same buffer are in UTF-8 encoding.

Configuring System Level Resources

Oracle SALT defines a set of global resources shared by all GWWS servers in the `SALTDEPLOY` file. The following system level resources can be configured in the `SALTDEPLOY` file:

- Certificates
- Plug-in load libraries

Configuring Certificates

Certificate information must be configured in order for the GWWS server to create an SSL listen endpoint, or to use X.509 certificates for authentication and/or message signature. All GWWS servers defined in the same deployment file shares the same certificate settings, including the private key file, trusted certificate directory, and so on.

The private key file is configured using the `<Certificate>/<PrivateKey>` sub-element. The private key file must be in PEM file format and stored locally.

SSL clients can optionally be verified if the `<Certificate>/<VerifyClient>` sub-element is set to `true`. By default, the GWWS server does not verify SSL clients.

If SSL clients are to be verified, and/or the X.509 certificate authentication feature is enabled, a set of trusted certificates must be stored locally and located by the GWWS server. There are two ways to define GWWS server trusted certificates:

1. Include all certificates in one PEM format file and define the file path using the `<<Certificate>/<TrustedCert>` sub-element.
2. Saving separate certificate PEM format files in one directory and define the directory path using the `<<Certificate>/<CertPath>` sub-element.

[Listing 11](#) shows a `SALTDEPLOY` file segment configuring GWWS server certificates.

Listing 11 Configuring Certificates In the SALTDEPLOY File

```
<Deployment ...>
...
<System>
  <Certificates>
    <PrivateKey>/home/user/gwws_cert.pem</PrivateKey>
    <VerifyClient>true</VerifyClient>
    <CertPath>/home/user/trusted_cert</CertPath>
```

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

Configuring Plug-in Libraries

A plug-in is a set of functions that are called when the GWSS server is running. Oracle SALT provides a plug-in framework as a common interface for defining and implementing plug-ins. Plug-in implementation is carried out through a dynamic library that contains the actual function code. The implementation library can be loaded dynamically during GWSS server start up. The functions are registered as the implementation of the plug-in interface.

In order for the GWSS server to load the library, the library must be specified using the <Plugin>/<Interface> element in the SALTDEPLOY file.

[Listing 12](#) shows a SALTDEPLOY file segment configuring multiple customized plug-in libraries to be loaded by the GWSS servers.

Listing 12 Configuring Plug-in Libraries In the SALTDEPLOY File

```
<Deployment ...>
...
<System>
  <Plugin>
    <Interface lib="plugin_1.so" />
    <Interface lib="plugin_2.so" />
  </Plugin>
</System>
</Deployment
```

Note: If the plug-in library is developed using the SALT 2.0 plug-in interface, the “id” and “name” attributes for the interface do not need to be specified. These values can be obtained through plug-in interfaces.

For more information, see [Using Plug-ins with Oracle SALT](#) in Oracle SALT Programming with Web Services.

Configuring Advanced Web Service Messaging Features

Oracle SALT currently supports the following advanced Web Service Messaging features:

- [Web Service Addressing](#)

Supports both inbound and outbound asynchronous Web service messaging.

- [Web Service Reliable Messaging](#)

Supports inbound Web Service reliable message delivery.

Web Service Addressing

Oracle SALT supports Web service addressing for both inbound and outbound services. The Web service addressing (WS-Addressing) messages used by the GWWS server must comply with the [Web Service Addressing standard \(W3C Member Submission 10 August 2004\)](#).

Inbound services do not require specific Web service addressing configuration. The GWWS server accepts and responds accordingly to both WS-Addressing request messages and non WS-Addressing request messages.

Outbound services require Web service addressing configuration as described in the following sections:

- [Configuring the Addressing Endpoint for Outbound Services](#)

- [Disabling WS-Addressing](#)

Configuring the Addressing Endpoint for Outbound Services

For outbound services, Web service addressing is configured at the Web service binding level. In the SALTDEPLOY file, each GWWS server can specify a WS-Addressing endpoint by using the <WSAddressing> element for any referenced outbound WSBinding object to enable WS-Addressing.

Once the WS-Addressing endpoint is configured, the GWWS server creates a listen endpoint at start up. All services defined in the outbound WSBinding are invoked with WS-Addressing messages.

[Listing 13](#) shows a SALTDEPLOY file segment enabling WS-Addressing for a referenced outbound Web service binding.

Listing 13 WS-Addressing Endpoint Defined for Outbound Web Service Binding

```
<Deployment ..>
  ...
  <WSGateway>
    <GWInstance id="GWWS1">
      ...
      <Outbound>
        <Binding ref="app1:app1_binding">
          <WSAddressing>
            <Endpoint address="https://myhost:8801/app1_async_point">
          </WSAddressing>
          <Endpoint use=" simpapp_GWWS1_HTTPPort" />
          <Endpoint use=" simpapp_GWWS1_HTTPSPort" />
        </Binding>
        <Binding ref="app2:app2_binding">
          <WSAddressing>
            <Endpoint address="https://myhost:8802/app2_async_point">
          </WSAddressing>
          <Endpoint use=" simpapp_GWWS1_HTTPPort" />
          <Endpoint use=" simpapp_GWWS1_HTTPSPort" />
        </Binding>
      </Outbound>
      ...
    </GWInstance>
  </WSGateway>
  ...
</ Deployment>
```

Notes: In a GWWS server, each outbound Web Service binding can be associated with a particular WS-Addressing endpoint address. These endpoints can be defined with the same hostname and port number, but the context path portion of the endpoint addresses must be different.

If the external Web service binding does not support WS-Addressing messages, configuring Addressing endpoints may result in run time failure.

Disabling WS-Addressing

No matter you create a WS-Addressing endpoint or not in the SALTDEPLOY file, you can explicitly disable the Addressing capability for particular outbound services in the WSDL. To disable the Addressing capability for a particular outbound service, you should use the property name “disableWSAddressing” with a value set to “true” in the corresponding <Service> definition in the WSDL file. This property has no impact to any inbound services.

[Listing 14](#) shows WSDL file segment disabling Addressing capability.

Listing 14 Disabling Service Level WS-Addressing

```
<Definition ...>
  <WSBinding id="simpapp_binding">
    <Servicegroup id="simpapp">
      <Service name="toupper">
        <Property name="disableWSAddressing" value="true" />
      </Service>
      <Service name="tolower" />
    </Servicegroup>
    ....
  </WSBinding>
</Definition>
```

Web Service Reliable Messaging

Oracle SALT currently supports Reliable Messaging for inbound services only. To enable Reliable Messaging functionality, you must create a Web Service Reliable Messaging policy file and include the policy file in the WSDL. The policy file must comply with the [WS-ReliableMessaging Policy Assertion Specification \(February 2005\)](#).

Note: A WSDL containing a Reliable Messaging policy definition should be used by the GWSS server for inbound direction only.

Creating the Reliable Messaging Policy File

A Reliable Messaging Policy file is a general WS-Policy file containing WS-ReliableMessaging Assertions. The WS-ReliableMessaging Assertion is an XML segment that describes features

such as the version of the supported WS-ReliableMessage specification, the source endpoint's retransmission interval, the destination endpoint's acknowledge interval, and so on.

For more information about the WS-ReliableMessaging policy file format, see the [Oracle SALT WS-ReliableMessaging Policy Assertion Reference](#) in the *Oracle SALT Reference Guide*.

[Listing 15](#) shows a Reliable Messaging policy file example.

Listing 15 Reliable Messaging Policy File Example

```
<?xml version="1.0"?>
<wsp:Policy wsp:Name="ReliableSomeServicePolicy"
  xmlns:wsm="http://schemas.xmlsoap.org/ws/2005/02/rm/policy"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:beapolicy="http://www.bea.com/wsm/policy">
  <wsm:RMAssertion>
    <wsm:InactivityTimeout Milliseconds="600000" />
    <wsm:AcknowledgementInterval Milliseconds="2000" />
    <wsm:BaseRetransmissionInterval Milliseconds="500"/>
    <wsm:ExponentialBackoff />
    <beapolicy:Expires Expires="P1D" />
    <beapolicy:QOS QOS="ExactlyOnce InOrder" />
  </wsm:RMAssertion>
</wsp:Policy>
```

Specifying the Reliable Messaging Policy File in the WSDL File

You must reference the WS-ReliableMessaging policy file at the <Servicegroup> level in the native WSDL file. The following segment of the WSDL file shows how to reference the WS-ReliableMessaging policy file.

Listing 16 Reference the WS-ReliableMessaging Policy At the Endpoint Level

```
<Definition ...>
  <WSBinding ...>
    <Servicegroup ...>
      <Policy location="RMPolicy.xml" />
    </Servicegroup ...>
  </WSBinding ...>
</Definition ...>
```



```

    <Service ... />
    <Service ... />
    ...
  </Servicegroup ...>
</WSBinding>
</Definition>

```

Note: Reliable Messaging in Oracle SALT does not support process/system failure scenarios, which means SALT does not store the message in a persistent storage area. Oracle SALT works in a *direct mode* with the SOAP client. Usually, system failure recovery requires business logic synchronization between the client and server.

Configuring Security Features

Oracle SALT provides security support at both transport level and SOAP message level. The following topics explain how to configure security features for each level:

- [Configuring Transport Level Security](#)
- [Configuring Message Level Web Service Security](#)

Configuring Transport Level Security

Oracle SALT provides point-to-point security using SSL link-level security and supports HTTP basic authentication mechanism for both inbound and outbound service authentication.

Setting Up SSL Link-Level Security

To set up link-level security using SSL at inbound endpoints, you can simply specify the endpoint address with prefix “https://”. The GWWS server who uses this inbound endpoint creates SSL listen port and make SSL secured connections with Web Service Clients. SSL features need to specify certificates settings. For more information about certificate settings, see “[Configuring Certificates](#)”.

GWWS server automatically creates SSL secured connection to outbound endpoints that are published with URLs that having prefix “https://”.

Configuring Inbound HTTP Basic Authentication

Oracle SALT depends on the Oracle Tuxedo security framework for Web Service client authentication. There is no special configuration at Oracle SALT side to enable inbound HTTP Basic Authentication. If the Oracle Tuxedo system requires user credential, HTTP Basic Authentication is simply an alternative for Web Service client program to carry the user credential.

The GWWS gateway supports Oracle Tuxedo domain security configuration for the following two authentication patterns:

- Application password (APP_PW)
- User-level authentication (USER_AUTH)

The GWWS server passes the following string from the HTTP header of the client SOAP request for Oracle Tuxedo authentication.

```
Authorization: Basic <base64Binary of username:password>
```

The following is an example of a string from the HTTP header:

```
Authorization: Basic QWxhZGRpbjpvYVUuIHNlc2FtZQ==
```

In this example, the client sends the Oracle Tuxedo username “Aladdin” and the password “open sesame”, and uses this paired value for Oracle Tuxedo authentication.

- Using Application Password (APP_PW)

If Oracle Tuxedo uses APP_PW, then the HTTP username value is ignored and the GWWS server only uses the password string as the Oracle Tuxedo application password to check the authentication.

- Using User-level Authentication (USER_AUTH)

If Oracle Tuxedo uses USER_AUTH, then both the HTTP username and password value are used. In this case, the GWWS server does not check the Oracle Tuxedo application password.

Note: ACL and MANDATORY_ACL are not supported for Web service clients, which means the Oracle Tuxedo system ignores any ACL-related configuration specifications. Oracle SALT does not make group information available for Web service clients.

Configuring Outbound HTTP Basic Authentication

Oracle SALT supports customers to develop authentication plug-in to prepare the user credential for the outbound HTTP Basic Authentication. Outbound HTTP Basic Authentication is configured at Endpoint level. If an outbound Endpoint requires user profile in the HTTP message,

you must specify the HTTP Realm for the HTTP endpoint in the WSDL file. The GWSS server invokes authentication plug-in library to prepare the username and password, and send them using HTTP Basic Authentication mechanism in the request message.

[Listing 17](#) shows how to enable HTTP Basic Authentication for the outbound endpoints.

Listing 17 Enabling HTTP Basic Authentication For the Outbound Endpoint

```
<Definition ...>
  <WSBinding id="simpapp_binding">
    <SOAP>
      <AccessingPoints>
        <Endpoint id="..." address="...">
          <Realm>SIMP_REALM</Realm>
        </Endpoint>
      </AccessingPoints>
    </SOAP>
    <Servicegroup id="simpapp">
      ....
    </Servicegroup>
    ....
  </WSBinding>
  .....
</Definition>
```

Once a service request is sending to an outbound endpoint specified with `<Realm>` setting, the GWSS server passes the Oracle Tuxedo client `uid` and `gid` to the authentication plug-in function, so that the plug-in can determine HTTP Basic Authentication `username/password` according to the Oracle Tuxedo client information. To obtain Oracle Tuxedo client `uid` / `gid` for HTTP basic authentication `username/password` mapping, Oracle Tuxedo security level may also need to be configured in the `UBBCONFIG` file. For more information, see [“Configuring Oracle Tuxedo Security Level for Outbound HTTP Basic Authentication”](#).

For more information about how to develop an outbound authentication plug-in, see [Programming Outbound Authentication Plug-ins](#) in the *Oracle SALT Programming Web Services*.

Configuring Message Level Web Service Security

Oracle SALT supports Web Service Security 1.0 and 1.1 specification for message level security. You can use message-level security in Oracle SALT to assure:

- Authentication, by requiring username or X.509 tokens
- Inbound request message integrity, by requiring the soap body signature

Main Use Cases of Web Service Security

Oracle SALT implementation of the *Web Service Security: SOAP Message Security specification* supports the following use cases:

- Include a token (username, or X.509) in the SOAP message for authentication.
- Include a token (X.509) and the soap body signature in the SOAP message for integrity.

Using WS-Security Policy Files

Oracle SALT includes a number of WS-Security Policy 1.0 and 1.2 files you can use for message level security use cases.

The WS-Policy files can be found at `$TUXDIR/udataobj/salt/policy` once you have successfully installed Oracle SALT.

[Table 8](#) lists the default WS-Security Policy files bundled by Oracle SALT.

Table 8 WS-Security Policy Files Provided By Oracle SALT

File Name	Purpose
wssp1.0-username-auth.xml	WS-Security Policy 1.0. Plain Text Username Token for Service Authentication
wssp1.0-x509v3-auth.xml	WS-Security Policy 1.0. X.509 V3 Certificate Token for Service Authentication
wssp1.0-signbody.xml	WS-Security Policy 1.0. Signature on SOAP:Body for verification of X.509 Certificate Token
wssp1.2-Wss1.0-UsernameToken-plain-auth.xml	WS-Security Policy 1.2. Plain Text Username Token for Service Authentication

Table 8 WS-Security Policy Files Provided By Oracle SALT

File Name	Purpose
wssp1.2-wss1.1-x509 V3-auth.xml	WS-Security Policy 1.2. X.509 V3 Certificate Token for Service Authentication
wssp1.2-signbody.xml	WS-Security Policy 1.2. Signature on SOAP:Body for verification of X.509 Certificate Token

The above policy files except WS-Security Policy 1.2 `UserToken` file can be referenced at `<Servicegroup>` or `<Service>` level in the native WSDL file. The WSSP 1.2 `UserToken` file can only be referenced at `<Servicegroup>` level. The sample "wsseapp" shows how to clip the WSSP 1.2 `UserToken` file used in `<Service>` level.

[Listing 18](#) shows a combination of policy assignment making that the service "TOUPPER" requires client send a `UsernameToken` (in plain text format) and an `X509v3Token` in request, and also require the `SOAP:Body` part of message is signed with the X.509 token.

Listing 18 WS-Security Policy Usage

```
<Definition ...>
  <WSBinding id="simpapp_binding">

    <Servicegroup id="simpapp">
      <Policy location="salt:wssp1.2-wss1.1-x509v3-auth.xml"/>
      <Service name="TOUPPER" >
        <Policy location="D:/wsseapp/wssp1.2-UsernameToken-Plain.xml"/>
        <Policy location="salt:wssp1.2-signbody.xml" use="input"/>
      </Service>
    </Servicegroup>

    ....
  </WSBinding>
  .....
</Definition>
```

Policy is referred with “location” attribute of the <Policy> element. A prefix “salt:” means an Oracle SALT default bundled policy file is used. User-defined policy file can be used by directly specifying the file path.

Notes: If a policy is referred at <Servicegroup> level, it will apply to all services in this service group.

The “signbody” policy must be used with the attribute “use” set as “input”, which specifies the policy applied only for input message. This is necessary because we do not sign the SOAP:Body of output message.

Compiling SALT Configuration

Compiling a SALT configuration file means generating a binary version of the file (SALTCONFIG) from the XML version SALTDEPLOY file. To compile a configuration file, run the `wsloadcf` command. `wsloadcf` parses a deployment file and loads the binary file.

`wsloadcf` reads a deployment file and all imported WSDL files and WS-Policy files referenced in the deployment file, checks the syntax according to the XML schema of each file format, and optionally loads a binary configuration file called SALTCONFIG. The SALTCONFIG and (optionally) SALTOFFSET environment variables point to the SALTCONFIG file and (optional) offset where the information should be stored.

`wsloadcf` validates the given SALT configuration files according to the predefined XML Schema files. XML Schema files needed by Oracle SALT can be found at directory: `$TUXDIR/udataobj/salt`.

`wsloadcf` can execute for validating purpose only without generating the binary version SALTCONFIG once option “-n” is specified.

For more information about `wsloadcf`, see [wsloadcf](#) reference in the *Oracle SALT Reference Guide*.

Configuring the UBBCONFIG File for Oracle SALT

After configuring and compiling the Oracle SALT configuration, the Oracle Tuxedo UBBCONFIG file needs to be updated to apply Oracle SALT components in the Oracle Tuxedo application.

[Table 9](#) lists the UBBCONFIG file configuration tasks for Oracle SALT.

Table 9 UBBCONFIG File Configuration Tasks for Oracle SALT

Configuration Tasks	Required	Optional
Configuring the TMMETADATA Server in the *SERVERS Section	X	
Configuring the GWWS Servers in the *SERVERS Section	X	
Updating System Limitations in the UBBCONFIG File	X	
Configuring Certificate Password Phrase For the GWWS Servers		X
Configuring Oracle Tuxedo Authentication for Web Service Clients		X
Configuring Oracle Tuxedo Security Level for Outbound HTTP Basic Authentication		X

Configuring the TMMETADATA Server in the *SERVERS Section

Oracle SALT requires at least one TMMETADATA server defined in the UBBCONFIG file. Multiple TMMETADATA servers are also allowed to increase the throughput of accessing the Oracle Tuxedo service definitions.

[Listing 19](#) lists a segment of the UBBCONFIG file that shows how to define TMMETADATA servers in a Oracle Tuxedo application.

Listing 19 TMMETADATA Servers Defined In the UBBCONFIG File *SERVERS Section

```

.....
*SERVERS
TMMETADATA SRVGRP=GROUP1 SRVID=1
           CLOPT="-A -- -f domain_repository_file -r"
TMMETADATA SRVGRP=GROUP1 SRVID=2
           CLOPT="-A -- -f domain_repository_file"
.....

```

Note: Maintaining only one Service Metadata Repository file for the whole Oracle Tuxedo domain is highly recommended. To ensure this, multiple TMMETADATA servers running in the Oracle Tuxedo domain must point to the same repository file.

For more information, see “[Managing The Tuxedo Service Metadata Repository](#)” in the *Tuxedo 9.1 documentation*.

Configuring the GWWS Servers in the *SERVERS Section

To boot GWWS instances defined in the SALTDEPLOY file, the GWWS servers must be defined in the *SERVERS section of the UBBCONFIG file. You can define one or more GWWS server instances concurrently in the UBBCONFIG file. Each GWWS server must be assigned with a unique instance id with the option “-i” within the Oracle Tuxedo domain. The instance id must be present in the XML version SALTDEPLOY file and the generated binary version SALTCONFIG file.

[Listing 20](#) lists a segment of the UBBCONFIG file that shows how to define GWWS servers in a Oracle Tuxedo application.

Listing 20 GWWS Servers Defined In the UBBCONFIG File *SERVERS Section

```
.....
*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 -- -c saltconf_2.xml -i GW3"
.....
```

For more information, see “[GWWS](#)” in the Oracle SALT Reference Guide.

Notes: Be sure that the TMMETADATA system server is set up in the UBBCONFIG file to start before the GWWS server boots. Because the GWWS server calls services provided by TMMETADATA, it must boot after TMMETADATA.

To ensure TMMETADATA is started prior to being called by the GWWS server, put TMMETADATA before GWWS in the UBBCONFIG file or use SEQUENCE parameters in *SERVERS definition in the UBBCONFIG file.

Oracle SALT configuration information is pre-compiled with `wsloadcf` to generated a binary version SALTCONFIG file. GWWS server reads SALTCONFIG file at start up.

Environment variable `SALTCONFIG` must be set correctly with the binary version `SALTCONFIG` file entity before booting `GWWS` servers.

Option “-c” is deprecated in the current version Oracle SALT. In SALT 1.1 release, option “-c” is used to specify SALT 1.1 configuration file for the `GWWS` server. In SALT 2.0, `GWWS` server reads `SALTCONFIG` file at start up. `GWWS` server specified with this option can be booted with a warning message to indicate this deprecation. The specified file can be arbitrary and is not read by the `GWWS` server.

Updating System Limitations in the `UBBCONFIG` File

When configuring the Oracle Tuxedo domain with SALT `GWWS` servers, you need to plan and update Oracle Tuxedo system limitations defined in the `UBBCONFIG` file according to your Oracle SALT application requirements.

Tip: Defining enough `MAXSERVERS` number in the `*RESOURCES` section

Oracle SALT requires the following system servers to be started in an Oracle Tuxedo domain: `TMMETADATA` and `GWWS`. The number of `TMMETADATA` and `GWWS` server must be accounted for in the `MAXSERVERS` value.

Tip: Defining enough `MAXSERVICES` number in the `*RESOURCES` section

When the `GWWS` server working in the outbound direction, external `wsdl:operations` are mapped with Oracle Tuxedo services and advertised via the `GWWS` servers. The number of the advertised services by all `GWWS` servers must be accounted for in the `MAXSERVICES` value.

Tip: Defining enough `MAXACCESSERS` number in the `*RESOURCES` section

`MAXACCESSERS` value is used to specify the default maximum number of clients and servers that can be simultaneously connected to the Oracle Tuxedo bulletin board on any particular machine in this application. The number of `TMMETADATA` and `GWWS` server, maximum concurrent Web Service client requests must be accounted for in the `MAXACCESSERS` value.

Tip: Defining enough `MAXWSCLIENTS` number in the `*MACHINES` section

When the `GWWS` server working in the inbound direction, each Web Service client is deemed a workstation client in Oracle Tuxedo system; therefore, `MAXWSCLIENTS` must be configured with

a valid number in `UBBCONFIG` for the machine where the `GWWS` server is deployed. The number shares.

Configuring Certificate Password Phrase For the `GWWS` Servers

Configuring security password phrase is required when setting up certificates for Oracle SALT. Certificates setting is desired when the `GWWS` servers enabling SSL link-level encryption and/or Web Service Security X.509 Token and signature features. The certificate private key file needs to be created and encrypted with a password phrase.

When the `GWWS` servers are specified with certificate related features, they are required to read the private key file and decrypt them using the password phrase. To configure password phrase for each `GWWS` server, keyword `SEC_PRINCIPAL_NAME` and `SEC_PRINCIPAL_PASSVAR` must be specified under each desired `GWWS` server entry in the `*SERVERS` section. During compiling the `UBBCONFIG` file with `tmlloadcf`, the administrator must type the password phrase, which can be used to decrypt the private key file correctly.

Note: Only one private key file can be specified in the Oracle SALT deployment file. All the `GWWS` servers defined in the Oracle SALT deployment file must be provided the same password phrase for the private key file decryption.

[Listing 21](#) lists a segment of the `UBBCONFIG` file that shows how to define security password phrase for the `GWWS` servers.

Listing 21 Security Password Phrase Defined in the `UBBCONFIG` File For the `GWWS` Servers

```
.....
*SERVERS
GWWS SRVGRP=GROUP1 SRVID=10
    SEC_PRINCIPAL_NAME="gwws_certkey"
    SEC_PRINCIPAL_VAR="gwws_certkey"
    CLOPT="-A -- -i GW1"
GWWS SRVGRP=GROUP1 SRVID=11
    SEC_PRINCIPAL_NAME="gwws_certkey"
    SEC_PRINCIPAL_PASSVAR="gwws_certkey"
    CLOPT="-A -- -i GW2"
.....
```

For more information, see “[UBBCONFIG\(5\)](#)” in the Oracle *Tuxedo 11gR1 documentation*.

Configuring Oracle Tuxedo Authentication for Web Service Clients

Oracle SALT GWSS servers rely on Oracle Tuxedo authentication framework to check the validity of the Web Service clients. If your legacy Oracle Tuxedo application is already applied with, Web Service clients must send user credential using one of the following approaches:

- HTTP Basic Authentication in the HTTP message header
- Web Service Security Username Token in the SOAP message header

Contrarily, if you want to authenticate Web Service clients for Oracle SALT, you must configure Oracle Tuxedo authentications in the Oracle Tuxedo domain.

For more information about Oracle Tuxedo authentication, see “[Administering Authentication](#)” in the *Oracle Tuxedo 11gR1 Documentation*.

Configuring Oracle Tuxedo Security Level for Outbound HTTP Basic Authentication

To obtain Oracle Tuxedo client `uid / gid` for outbound HTTP Basic Authentication `username /password` mapping, you need to configure Oracle Tuxedo Security level as `USER_AUTH`, `ACL` or `MANDATORY_ACL` in the `UBBCONFIG` file.

[Listing 22](#) lists a segment of the `UBBCONFIG` file that shows how to define security level `ACL` in the `UBBCONFIG` file.

Listing 22 Security Level ACL Defined in the UBBCONFIG File For Outbound HTTP Basic Authentication

```
*RESOURCES
IPCKEY ...
.....
SECURITY ACL
.....
```

Configuring Oracle SALT In Oracle Tuxedo MP Mode

To set up GWSS servers running on multiple machines within a MP mode Oracle Tuxedo domain, each Oracle Tuxedo machine must be defined with a separate `SALTDEPLOY` file and a set of separate other components.

You must propagate the following global resources across different machines:

- Certificates.

Private key file and the trusted certificate files must be accessible from each machine according to the settings defined in the `SALTDEPLOY` file.

- Plug-in load libraries.

Plug-in shared libraries must be compiled on each machine and must be accessible according to the settings defined in the `SALTDEPLOY` file.

You may define two GWSS servers running on different machine with the same functionality by associating the same WSDF files. But it requires manual propagation of the following artifacts:

- The WSDF files
- The WS-Policy files
- FML32 field table definition files if Oracle Tuxedo Services consume FML32 typed buffers
- XML Schema files excerpted by `wsdlcvt`.

Migrating from Oracle SALT 1.1

This section describes the following two possible migrating approaches for SALT 1.1 customers who plan to upgrade to SALT 2.0 release:

- [Running GWSS servers with SALT 1.1 Configuration File](#)
- [Adopting SALT 2.0 Configuration Style by Converting SALT 1.1 Configuration File](#)

Running GWSS servers with SALT 1.1 Configuration File

After upgrading from SALT 1.1 to SALT 2.0 release, you may still want to run your existing SALT applications with the original SALT 1.1 configuration file. SALT 2.0 definitely supports that.

SALT configuration compiler utility, `wsloadcf`, supports to load the binary version `SALTCONFIG` from one SALT 1.1 format configuration file.

To run SALT 2.0 `GWWS` servers with SALT 1.1 Configuration file, you need to perform the following steps:

1. Load the binary version `SALTCONFIG` from the SALT 1.1 format configuration file via `wsloadcf`.
2. Set environment variable `SALTCONFIG` before booting the `GWWS` servers.
3. Boot the `GWWS` servers associated with this SALT 1.1 configuration file.

Note: If customers have more than one SALT 1.1 configuration files defined in an Oracle Tuxedo domain, customers need to follow step 1 to 3 to generate more binary version `SALTCONFIG` files and boot corresponding `GWWS` servers.

Adopting SALT 2.0 Configuration Style by Converting SALT 1.1 Configuration File

When `wsloadcf` loads a binary version `SALTCONFIG` from a SALT 1.1 configuration file, it also convert this SALT 1.1 configuration file into one `WSDF` file and one `SALTDEPLOY` file.

It's highly recommended to start using the SALT 2.0 styled configuration once you get the converted files from SALT 1.1 configuration.

Note: If customers want to incorporate more than one SALT 1.1 configuration files into one SALT 2.0 deployment, customers need to manually edit the `SALTDEPLOY` file for importing the other `WSDF` files.

[Listing 23](#) lists the converted `SALTDEPLOY` file and `WSDF` file from a given SALT 1.1 configuration file.

Listing 23 A Sample of SALT 1.1 Configuration File (`simpapp.xml`)

```
<Configuration xmlns=" http://www.bea.com/Tuxedo/Salt/200606">
  <Servicelist id="simpapp">
    <Service name="toupper" />
    <Service name="tolower" />
  </Servicelist>
  <Policy />
  <System />
```

```

<WSGateway>
  <GWInstance id="GWWS1">
    <HTTP address="//127.0.0.1:7805" />
    <HTTPS address="127.0.0.1:7806" />
    <Property name="timeout" value="300" />
  </GWInstance>
</WSGateway>
</Configuration>

```

The converted SALT 2.0 WSDL file and deployment file are shown in [Listing 24](#) and [Listing 25](#).

Listing 24 Converted WSDL File for SALT 1.1 Configuration File (simpapp.xml.wsdl)

```

<Definition name="simpapp" wsdlNamespace="urn:simpapp.wsdl"
  xmlns=" http://www.bea.com/Tuxedo/WSDL/2007">
  <WSBinding id="simpapp_binding">
    <Servicegroup id="simpapp">
      <Service name="toupper" />
      <Service name="tolower" />
    </Servicegroup>
    <SOAP>
      <AccessingPoints>
        <Endpoint id="simpapp_GWWS1_HTTPPort"
          address=http://127.0.0.1:7805/simpapp />
        <Endpoint id=" simpapp_GWWS1_HTTPSPort"
          address=https://127.0.0.1:7806/simpapp />
      </AccessingPoints>
    </SOAP>
  </WSBinding>
</Definition>

```

Listing 25 Converted SALTDEPLOY File for SALT 1.1 Configuration File (simpapp.xml.dep)

```
<Deployment xmlns=" http://www.bea.com/Tuxedo/SALTDEPLOY/2007">
  <WSDF>
    <Import location="/home/myapp/simpapp.wsdf" />
  </ WSDF>
  <WSGateway>
    <GWInstance id="GWWS1">
      <Inbound>
        <Binding ref="simpapp:simpapp_binding">
          <Endpoint use=" simpapp_GWWS1_HTTPPort" />
          <Endpoint use=" simpapp_GWWS1_HTTPSPort" />
        </Binding>
      </Inbound>
      <Properties>
        <Property name="timeout" value="300" />
      </Properties>
    </GWInstance>
  </WSGateway>
</ Deployment>
```

Configuring Oracle Tuxedo SCA Components

Configuring Oracle Tuxedo SCA components comprises the following:

- [Configuring an SCA ATMI Client](#)
- [Configuring an SCA JATMI Client](#)
- [Configuring an SCA Workstation Client](#)
- [Configuring an SCA Web Service Client](#)
- [Configuring an SCA ATMI Server](#)
- [Configuring an SCA Web Service Server](#)
- [Configuring SCA Client Security](#)

Configuring an SCA ATMI Client

The SCA ATMI client is a native Oracle Tuxedo client that is written using the SCA paradigm and built using the `buildscaclient` utility. The client executable must be in a subdirectory of a directory that contains the `root.composite` file.

Note: The `APPDIR` environment variable must point to the `root.composite` file directory.

[Listing 26](#) shows the client application root composite file `$APPDIR/root.composite`.

Listing 26 Client Application Root Composite File

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0" name="testApp">
  <component name="testStringClientComp">
    <implementation.composite name="ECHO"/>
  </component>
</composite>
```

The `$APPDIR/ECHO` directory contains the `ECHO` application. The directory name, "ECHO", must match the name specified in `<implementation.composite name="ECHO"/>`. [Listing 27](#) shows the client application composite file.

Listing 27 Client Application Composite File

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0" name="ECHO">
  <reference name="ECHO">
    <interface.cpp header="ECHO.h"/>
    <binding.atmi requires="legacy">
      <tuxconfig>/tux/application/ECHOserver/tuxconfig</tuxconfig>
      <inputBufferType target="TestString">STRING</inputBufferType>
      <outputBufferType target="TestString">STRING</outputBufferType>
      <errorBufferType target="TestString">STRING</errorBufferType>
    </binding.atmi>
  </reference>
</composite>
```



```
</reference>
</composite>
```

The client dynamic link library for this client application is also contained in this directory. For example, using the example in [Listing 27](#), the `$APPDIR/ECHO/ECHO.so` shared object exists in the ECHO directory. The target "TestStr" is used to group buffer types together.

The client executable also exists in this directory. There is no naming convention associated with a client application. This client ECHO application could very well contain "doEchoClient" in the ECHO application directory. For example: `$APPDIR/ECHO/doEchoClient`.

Note: You must set `SCA_COMPONENT`. See [Listing 27](#),
`SCA_COMPONENT=testStringClientComp`.

Configuring an SCA JATMI Client

The JATMI client application configuration composite file is part of the Java `.jar` file. The JATMI client composite file is not part of any package and is located in the base of the `.jar` file. When client application is invoked, SCA Java runtime loads the composite file. No special setup is required.

Note: The client application `.jar` file *must* be included in the `CLASSPATH`. The following `.jar` files should also be part of `CLASSPATH`:

- `binding-jatmi-extension.jar`
- `com.oracle.jatmi.dataxfm_1.0.0.0.jar`
- `com.bea.core.jatmi_1.2.0.3.jar`
- `com.bea.core.i18n_1.4.0.0.jar`
- `tuscany-sca-manifest.jar`

[Listing 28](#) shows an SCA JATMI client composite file example.

Listing 28 SCA JATMI Client Composite File Example

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
xmlns:f="binding-atmi.xsd"
name="EchoComposite">
```

```

<reference name="ECHO" promote="EchoComponent/ECHO">
  <interface.java class="com.abc.sca.java.Echo" />
  <f:binding.atmi requires="legacy">
    <f:serviceType>RequestResponse</f:serviceType>
    <f:inputBufferType>FML</f:inputBufferType>
    <f:outputBufferType>FML</f:outputBufferType>
    <f:fieldTables>com.abc.sca.java.fml.FMLTABLE
  </f:fieldTables>
  <f:workStationParameters>
    <f:networkAddress>//STRIATUM:15011
  </f:networkAddress>
  </f:workStationParameters>
  </f:binding.atmi>
</reference>
<component name="EchoComponent">
  <implementation.java
    class="com.abc.sca.java.EchoComponentImpl" />
</component>
</composite>

```

Configuring an SCA Workstation Client

Configuring an SCA workstation clients is similar to configuring SCA native clients. One difference is that an SCA workstation client requires using the `<workStationParameters>` element and its sub-elements in the composite. The SCA runtime automatically detects whether the client is built as an SCA native client or SCA workstation client and loads the correct reference binding library accordingly.

An SCA Oracle Tuxedo Workstation client has a similar directory hierarchy to an SCA native client. Both rely on the environment variable `$APPDIR`, which points to where the client application is located.

[Listing 29](#) and [Listing 30](#) show SCA Oracle Tuxedo workstation client configuration examples.

Listing 29 \$APPDIR/root.composite

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0" name="testApp">
  <component name="testStringClientComp">
    <implementation.composite name="ECHO"/>
  </component>
</composite>
```

Listing 30 \$APPDIR/ECHO/ECHO.composite

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0" name="ECHO">
  <reference name="ECHO">
    <interface.cpp header="ECHO.h"/>
    <binding.atmi requires="legacy">
      <inputBufferType target="TestString">STRING</inputBufferType>
      <outputBufferType target="TestString">STRING</outputBufferType>
      <errorBufferType target="TestString">STRING</errorBufferType>
    <workStationParameters>
      <networkAddress>//STRIATUM:4890</networkAddress>
      <encryptBits>128/128</encryptBits>
    </workStationParameters>
    <remoteAccess>WorkStation</remoteAccess>
  </binding.atmi>
</reference>
</composite>
```

Configuring an SCA Web Service Client

The SCA Web service client is basically the same as SCA native client except that it uses the `<binding.ws>` element instead of `<binding.atmi>`. The SCA runtime automatically detects which binding the client is using, and loads the correct reference binding accordingly.

The SCA Web service client has a similar directory hierarchy as native client. They both rely on the \$APPDIR environment variable to point to where the client application is located.

[Listing 31](#) and [Listing 32](#) show SCA Web service client configuration examples.

Listing 31 \$APPDIR/root.composite

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.oesa.org/xmlns/sca/1.0" name="testApp">
  <component name="calcClient">
    <implementation.composite name="calcClient"/>
  </component>
</composite>
```

Listing 32 \$APPDIR/calcClient/calcClient.composite

```
<composite xmlns="http://www.oesa.org/xmlns/sca/1.0" name="calcClient">
  <reference name="Calculator">
    <interface.cpp header="CalcService.h"/>
    <binding.ws
      endpoint="http://calc.sample#wsdl.endpoint
        (Calculator/CalculatorSOAP11port)"/>
  </reference>
</composite>
```

The `<interface.cpp>` element is required to build the appropriate proxy stub. Also, the client directory should contain the WSDL file where the endpoint specified in `<binding.ws>` is located. In addition, the configuration of the Oracle Tuxedo Web services gateway (GWWS) is necessary and requires the following steps:

1. Make sure the TMMETADATA and GWWS servers are shut down.
2. Run `wSDLcvt` on the WSDL of the service(s) used. This produces a WSDL file, an Oracle Tuxedo Metadata Repository interface definitions file, fml32 field tables and XML schemas.

3. Optionally, modify the generated WSDL file to override the actual endpoint address used at runtime. For more information, see WSDL documentation.
4. Load the Oracle Tuxedo Metadata Repository interface definitions into the TMMETADATA server repository (e.g.: `$ tmlloadrepos -I calc.mif metadata.repos -y`). For more information, see `tmlloadrepos` documentation.
5. Add a reference to the WSDL in the GWWS configuration input file (named `gwws.dep` for example). [Listing 33](#) shows the added elements highlighted in blue.
6. Reload the GWWS binary configuration file to take into account the changes performed in the previous five (e.g.: `$ wsloadcf -y gwws.dep`).
7. Reboot GWWS and TMMETADATA.

Listing 33 GWWS Configuration File

```

<?xml version="1.0" encoding="UTF-8"?>
<saltdep:Deployment
xmlns:saltdep="http://www.bea.com/Tuxedo/SALTDEPLOY/2007"
xmlns="http://www.bea.com/Tuxedo/SALTDEPLOY/2007"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <saltdep:WSDF>
    <saltdep:Import location="calc.wsdf"/>
  </saltdep:WSDF>
  <saltdep:WSGateway>
    <saltdep:GWInstance id="GWWS1">
      <saltdep:Outbound>
        <saltdep:Binding ref="calc:CalculatorSOAP11Binding">
          <saltdep:Endpoint use="CalculatorSOAP11port"/>
        </saltdep:Binding>
      </saltdep:Outbound>
    </saltdep:GWInstance>
  </saltdep:WSGateway>
  <saltdep:System/>
</saltdep:Deployment>

```

Configuring an SCA ATMI Server

For an SCA ATMI server, the SCA ROOT is the same as `$APPDIR`. There should be at least one composite file that describes the SCA application. The SCA runtime searches for this composite file and from there it loads all the `composite` and `componentType` files for SCA server applications that are hosted in an Oracle Tuxedo environment.

[Listing 34](#) shows a root composite file, named `root.composite` contains two SCA applications hosted in an Oracle Tuxedo application domain. The two applications are called Purchase and Finance. There are at least two subdirectories for these two SCA applications. One is called `Purchase.component` and the other is called `Finance.component`.

Listing 34 `$APPDIR/root.composite`

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0" name="root">
  <component name="Purchase.component">
    <implementation.composite name="Purchase" />
  </component>
  <component name="Finance.component">
    <implementation.composite name="Finance" />
  </component>
</composite>
```

[Listing 35](#) shows the `Purchase.component` directory contains a composite file for the Purchase application named `Purchase.composite`. Similarly, the `Finance.component` directory contains a composite file for the Finance application named `Finance.composite`.

Listing 35 `$APPDIR/Purchase.component/Purchase.composite`

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
  name="Purchase">
  <service name="purchase">
    <interface.cpp header="Purchase.h" />
  <binding.atmi requires="legacy">
```

```

        <map target="Order">ORDER</map>
        <map target="TrackOrder">TRACKORDER</map>
    </binding.atmi>
    <reference>PurchaseServiceComponent</reference>
</service>
    <component name="PurchaseServiceComponent">
        <implementation.cpp library="Purchase"
header="PurchaseImpl.h" />
    </component>
</composite>

```

Listing 36 shows `Purchase.composite` contains the `PurchaseImpl.componentType` file in the `$APPDIR/Purchase.component` directory and uses `CPP` as its application implementation. When an SCA server using this configuration is built using the `buildscaserver` utility, it advertises two SCA services automatically at runtime (`ORDER` and `TRACKORDER`). The actual `CPP` implementation of the services is called `Order` and `TrackOrder`.

Listing 36 `$APPDIR/Purchase.component/PurchaseImpl.componentType`

```

<?xml version="1.0" encoding="UTF-8"?>
<componentType xmlns="http://www.oxa.org/xmlns/sca/1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <service name="purchase">
        <interface.cpp header="Purchase.h"/>
    </service>
</componentType>

```

Assume these two SCA applications hosted in Oracle Tuxedo and built using `buildscaserver` are called `PurchaseSvr` and `FinanceSvr`. You must add the following lines to the `*SERVERS` section in the `UBBCONFIG` file:

```

PurchaseSvr SRVGRP=PURCHASEGRP SRVID=500
FinanceSvr SRVGRP=FINANCEGRP SRVID=600

```

There is no need to add a service in the *SERVICES section. SCA services hosted by Oracle Tuxedo are dynamically advertised.

Configuring an SCA Web Service Server

Configuring Web services binding for components (server side) is similar to configuring ATMI binding for hosting SCA components.

[Listing 37](#) shows a root composite file named `root.composite`. It contains one SCA component hosted in an Oracle Tuxedo application domain. The two applications are called Purchase and Finance. There are at least two subdirectories for these two SCA applications, one is called `Purchase.component`, and the other is called `Finance.component`.

[Listing 38](#) shows the actual component subdirectory. [Listing 39](#) shows the `componentType` side file

Listing 37 \$APPDIR/root.composite

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0" name="root">
  <component name="account">
    <implementation.composite name="account" />
  </component>
</composite>
```

Listing 38 \$APPDIR/account/account.composite

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
  name="account">
  <service name="AccountService">
    <interface.wsdl
interface="http://www.bigbank.com/AccountService#wsdl.interface(AccountSer
vice)"/>
    <binding.ws/>
    <reference>AccountServiceComponent</reference>
  </service>
```



```

    <component name="AccountServiceComponent">
        <implementation.cpp library="Account"
header="AccountServiceImpl.h"/>
    </component>
</composite>

```

Listing 39 \$APPDIR/account/AccountServiceImpl.componentType

```

<?xml version="1.0" encoding="UTF-8"?>
<componentType xmlns="http://www.osoa.org/xmlns/sca/1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <service name="AccountService">
        <interface.cpp header="AccountService.h"/>
    </service>
</componentType>

```

The above SCA component will be hosted in an Oracle Tuxedo server built using `buildscaserver` with the `-w` option (for Web services) and named `WSServer`

Then in the Oracle Tuxedo `UBBCONFIG` file you need to add the following line in the `*SERVERS` section: `WSServer SRVGRP=ACCTGRP SRVID=500`.

There is no need add a service in the `*SERVICES` section. SCA services hosted by Oracle Tuxedo are dynamically advertised.

In addition, configuration of the Oracle Tuxedo Web services gateway (GWWS) is necessary. Do the following steps:

1. Make sure the `TMMETADATA` and `GWWS` servers are shut down
2. Run `wsdlcvt` on the WSDL of the service(s) used. This produces a WSDL file, an Oracle Tuxedo Metadata Repository interface definitions file, `fml32` field tables and XML schemas.
3. Modify the generated WSDL file to specify the actual endpoint address used at runtime to accept requests. For more information, see WSDL documentation.

4. Load the Oracle Tuxedo Metadata Repository interface definitions into the TMMETADATA server repository (e.g.: \$ `tmloadrepos -I AccountService.mif metadata.repos -y`). For more information, see `tmloadrepos` documentation.
5. Add a reference to the WSDF in the GWWS configuration input file (named `gwws.dep` for example). [Listing 40](#) shows the elements added highlighted in blue.
6. Reload the GWWS binary configuration file to take into account the changes performed in the step five (e.g.: \$ `wsloadcf -y gwws.dep`).
7. Reboot GWWS and TMMETADATA.

Listing 40 gwws.dep File

```
<?xml version="1.0" encoding="UTF-8"?>
<saltdep:Deployment
xmlns:saltdep="http://www.bea.com/Tuxedo/SALTDEPLOY/2007"
xmlns="http://www.bea.com/Tuxedo/SALTDEPLOY/2007"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <saltdep:WSDF>
    <saltdep:Import location="AccountService.wsdf"/>
  </saltdep:WSDF>
  <saltdep:WSGateway>
    <saltdep:GWInstance id="GWWS1">
      <saltdep:Inbound>
        <saltdep:Binding
ref="AccountService:AccountServiceSOAP">
          <saltdep:Endpoint use="AccountServiceSOAP"/>
        </saltdep:Binding>
      </saltdep:Inbound>
    </saltdep:GWInstance>
  </saltdep:WSGateway>
  <saltdep:System/>
</saltdep:Deployment>
```

Configuring SCA Client Security

Oracle Tuxedo SCA components support two types of security:

- [Oracle Tuxedo Application Domain Security](#)
- [Oracle Tuxedo Link-Level Security](#)

Oracle Tuxedo Application Domain Security

Oracle Tuxedo Application Domain Security is set when the `TUXCONFIG` file for the Oracle Tuxedo Application Domain contains the `SECURITY` keyword in the `*RESOURCES` section. There are five levels of application security: `NONE`, `APP_PW`, `USER_PW`, `ACL`, and `MANDATORY_ACL`. All security levels except `NONE` require at least an application password from user to gain access to the Oracle Tuxedo application. At the `USER_PW` level and above there is an additional user password to authenticate the user and establish user credentials. In total there are potentially two passwords that need to be configured.

All SCA clients require this password information in order to gain access to Oracle Tuxedo application servers. There are two ways for an SCA client to retrieve password information:

- The client application may provide password information to ATMI/JATMI reference binding extensions through a callback mechanism.
- The client application may configure the identification of the password to be retrieved by the ATMI/JATMI reference binding extensions in the appropriate composite file.

Note: For more information, see Password callback methods in the [Oracle SALT Programmer's Guide](#).

In order for the Oracle SALT administrator to configure password retrieval, the administrator must:

- Maintain the `password.store` file and set this file up correctly for the client application. The administrator must duplicate the `password.store` file across different machines if necessary.
- Add or delete password and identification pairs when necessary.
- Configure the client application composite file with correct information.

[Listing 41](#) and [Listing 42](#) contain SCA ATMI client application examples.

Listing 41 \$APPDIR/password.store \$APPDIR/simple.app.composite

```
<?xml version="1.0" encoding="UTF-8"?>

<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
  name="simple.app">

  <component name="simpapp.TuxClient">
    <implementation.composite name="simpapp.client"/>
    <reference name="TOUPPER">tuxToupper</reference>
  </component>

</composite>
```

Listing 42 \$APPDIR/simpapp.client/simpapp.client.composite

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
  name="simpapp.client">

  <reference name="TOUPPER">
    <interface.cpp header="ToupperTuxService.h"/>
    <binding.atmi requires="legacy">
      <tuxconfig>d:\tests\tuxedo\sca\tests\TUXCONFIG</tuxconfig>
      <inputBufferType target="charToup">STRING</inputBufferType>
      <outputBufferType target="charToup">STRING</outputBufferType>
      <authentication
        <passwordIdentifier>aaa</passwordIdentifier>
      </authentication>
    </binding.atmi>
  </reference>
</composite>
```

The above composite defines an Oracle Tuxedo application domain password identification "aaa" which will cause the ATMI reference binding to retrieve the password with identification

"aaa" from the `password.store` file at the runtime. If you increased Oracle Tuxedo application domain security by requiring user authentication. (`SECURITY=USER_PW` or above) you would use the following command: `scapasswordtool -i crusoe -a`.

Then use a text editor or any other tool that can edit the `simpapp.client.composite` file and add the following entry in the `<binding.atmi/authentication>` element:

```
<userPasswordIdentifier>crusoe</userPasswordIdentifier>
```

Anyone using the password "crusoe" can access Oracle Tuxedo applications.

Oracle Tuxedo Link-Level Security

Oracle Tuxedo Link-Level Security has two variations. One is the easily configured Link-Level Encryption (LLE) and the other one is the more commonly used Transport Layer Security (TLS) also known as Secured Socket Layer (SSL). An SCA ATMI client using the native ATMI reference binding does not need link-level security configured at the SCA level since its transport method is native message queues and the Oracle Tuxedo BRIDGE.

The SCA JATMI client reference binding does not support link-level security. The only type of SCA client that allows configuration of link-level security is SCA Workstation ATMI client.

The SCA Workstation ATMI client contains a `<workStationParameters>` element configured in the composite file. The SCA runtime automatically loads the correct reference binding for this type of client.

Configuring Link-Level Encryption

Link-level encryption can be configured by adding an `<encryptBits>` element in the composite file. The following elements *should not* be configured for LLE, since they are specific to SSL encryption and imply that SSL encryption is used:

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

The `<encryptBits>` element specifies the encryption strength that this client will attempt to negotiate. The syntax for the `<encryptBits>` element is `<minimum encryption strength>/<maximum encryption strength>`. To configure minimum 56-bit encryption you must add the following to the composite file:

```
<networkAddress>//STRIATUM:8741</networkAddress>
<encryptBits>56/128</encryptBits>
```

Note: `encryptBits` specifies the encryption strength that the client connection attempts to negotiate. The format is `<minencryptbits>/<maxencprytbits>` (for example, 128/128). Values can be 0 (no encryption is used), 40, 56, 128, or 256. Invalid values result in a configuration exception.

This tells SCA Workstation Reference binding to require 56 to 128 bits encryption strength when negotiating with WSH. You must also add the following line to the `*SERVERS` section in the `UBBCONFIG` file:

```
WSL   SRVGRP=GROUP1 SRVID=1001 CLOPT="-A -- -n //STRIATUM:8741 -a -z 56 -Z
256
```

Configuring Transport Layer Security

In addition to `<encryptBits>`, to enable Link-Level Security over TLS/SSL you must configure `secPrincipalName`, `secPrincipalLocation`, and `secPrincipalPassId`.

- `secPrincipalName` - the name of the security principal. It is used for searching the client X.509 certification from the LDAP server.
- `secPrincipalLocation` - the client private key file.
- `secPrincipalPassId` - the password identifier that is used to retrieve client password used to encrypt the private key file.

These three parameters specify the parameters needed when a TLS/SSL connection needs to be established by a SCA Workstation ATMI client.

[Listing 43](#) contains the lines you must add to the client composite file in `/binding.atmi/workStationParameters` to configure TLS/SSL.

Listing 43 Client Composite File

```
<networkAddress>//STRITUM:8742</networkAddress>
<secPrincipalName>crusoe</secPrincipalName>
<secPrincipalLocation>/tux/udataobj/security/keys/crusoe.pem</secPrincipal
Location>
<secPrincipalPassId>crusoe</secPrincipalPassId>
```

In Oracle Tuxedo, you must add `-S 8742` to `WSL` to indicate that TLS/SSL is used if the client connects through port 8742.

```
WSL SRVGRP=GROUP1 SRVID=1001
CLOPT="-A -- -n //STRIATUM:8741 -S 8742 -z 56 -Z 128"
```

Configuring Service Contract Discovery

When discovery is activated for a service, the server that provides the service collects service contract information and sends the information to an internal service implemented by [TMMETADATA \(5\)](#). The same service contract is only sent once to reduce communication overhead.

The TMMETADATA server summarizes the collected data and generates a service contract. The contract information can either be stored in the metadata repository, or output to a text file (which is then loaded to the metadata repository using `tmloadrepos`). Oracle SALT uses the `tmscd` command to control the service contract runtime collection. For more information, see [tmscd](#) in the *Oracle SALT Command Reference Guide*.

Generated service contract information contains service name, request buffer information, response buffer information, and error buffer information if there is a failure. The collected service contract information is discarded if it fails to send information to the TMMETADATA server. The buffer information includes buffer type and subtype, and field information for FML/FML32 (`name, type, subtype`).

Discovery is supported for any embedded buffer in FML32 buffer. For FML/FML32 field occurrences, the discovery automatically updates the pattern for the `count/requiredcount` in metadata repository. Field occurrence does not impact pattern, but the minimum occurrence is the `"requiredcount"`. The maximum occurrence is the `"count"` of the entire contract discovery period.

For VIEW/VIEW32/X_C_TYPE/X_COMMON, only the view name is discovered. ORACLE SALT can generate view detail description by view name when using metadata repository.

Note: Patterns flagged with `autodiscovery` (see [Table 10](#)) are compared.

If the same autodiscovery pattern already exists in the metadata repository, then the newer pattern is ignored.

Only application ATMI services (local, or imported via /TDOMAIN gateway) are supported. Service contract discovery *does not* support the following services:

- system services (name starts with '.' or '..')
- conversational services

- CORBA services
- /Q and Oracle SALT proxy services

Note: Services without a reply are mapped as "oneway" services in the metadata repository.

tpforward Support

If a service issues `tpforward()` instead of `tpreturn()`, its reply buffer information will be same with the reply buffer of the service which it forwards to. For example,

- client calls SVCA with a STRING typed buffer
- SVCA processes the request, and then creates a new FML32 typed buffer as request forwarded to SVCB
- SVCB handles the request and returns a STRING buffer to the client. The SVCA contract is `STRING+STRING`. The SVCB contract is `FML32+STRING`

Service Contract Text File Output

If you want collected service contract discovery information logged to a file instead of directly to the metadata repository, you must use the `TMMETADATA(5) -o<filename>` option and then use `tmloadrepos` to manually load the file to the metadata repository. For more information, see `tmloadrepos` in the [Oracle Tuxedo Command Reference Guide](#).

The output complies with the format imposed by `tmloadrepos` if a file is used as storage instead of metadata repository. The file contains a service header section and a parameter section for each parameter. Service header contains items listed in [Table 10](#). The "service" field format is `<TuxedoServiceName>+'_' +<SequenceNo>`. The suffix `<SequenceNo>` is used to avoid name conflict when multiple patterns are recognized for an Oracle Tuxedo service.

Note: `<SequenceNo>` starts from the last `<SequenceNo>` number already stored in the metadata repository.

Service parameter contains information is listed in [Table 11](#).

Table 10 Service Level Attributes

Keyword (abbreviation)	Sample Value	Description
<code>service(sv)</code>	TOUPPER_1	<RealServiceName>_<SequenceNo>.
<code>tuxservice(tsv)</code>	TOUPPER	The service name.
<code>servicetype(st)</code>	<code>service oneway</code>	oneway if TPNOREPLY is set.
<code>inbuf(bt)</code>	STRING	FML, FML32, VIEW, VIEW32, STRING, CARRAY, XML, X_OCTET, X_COMMON, X_C_TYPE, MBSTRING or other arbitrary string representing an application defined custom buffer type.
<code>outbuf(BT)</code>	FML32	set to "NULL" if it's an error reply.
<code>errbuf(ebt)</code>	STRING	present only when it's an error reply.
<code>inview</code>		View name. Present only when inbuf is of type VIEW or VIEW32.
<code>outview</code>		View name. Present only when outbuf is of type VIEW or VIEW32.
<code>errview</code>		View name. Present only when errbuf is of type VIEW or VIEW32.
<code>autodiscovery</code>	true	Set to "true".

Table 11 Parameter Level Attributes

Keyword(abbreviation)	Sample	Description
param(pn)	USER_INFO	
paramdescription(pd)	service parameter	
access(pa)	in	A combination of {in}{out}{err}.
type(pt)	fml32	byte, short, integer, float, double, string, carray, dec_t, xml, ptr, fml32, view32, mbstring.
subtype(pst)		A view name for a view or view32 typed parameter.
count	100	The maximum occurrence of FML/FML32 field watched during the collection period
requiredcount	1	The minimum occurrence of FML/FML32 field watched during the collection period.

Examples

Example 1:

Input: service=SVC, request=STRING, reply = TPSUCCESS + STRING

Output Pattern: service=SVC_1,tuxservice=SVC,inbuf=STRING,outbuf=STRING

Example 2:

Input: service=SVC, request=STRING, reply = TPFAIL+ STRING

Output Pattern (partial): Service=SVC_1,
tuxservice=SVC,inbuf=STRING,outbuf=NULL,errbuf=STRING

Example 3:

Input:

service=SVC, request=STRING, reply = TPSUCCESS + STRING

```
service=SVC, request=STRING, reply = TPFAIL+ STRING
```

Output Pattern:

```
service=SVC_1,tuxservice=SVC,inbuf=STRING,outbuf=STRING
```

```
Service=SVC_2, tuxservice=SVC,inbuf=STRING,outbuf=NULL,errbuf=STRING
```

Example 4:

Input: `service=FMLS,request=FML32(name,pwd),reply=TPSUCCESS+FML32(id)`

Output Pattern:

```
service=FMLS_1,tuxservice=FMLS,inbuf=FML32,outbuf=FML32
```

```
param: input(name, pwd), output(id)
```

Example 5:

Input:

```
service=FMLS,request=FML32(name,pwd),reply=TPSUCCESS+FML32(id)
```

```
service=FMLS,request=FML32(name,pwd,token),reply=TPSUCCESS+FML32(id)
```

Output Pattern:

```
service=FMLS_1,tuxservice=FMLS,inbuf=FML32,outbuf=FML32
```

```
param: input(name, pwd), output(id)
```

```
service=FMLS_2,tuxservice=FMLS,inbuf=FML32,outbuf=FML32
```

```
param: input(name, pwd,token), output(id)
```

Configuring Oracle SALT WS-TX Support

This section contains the following topics:

- [Configuring Transaction Log Device](#)
- [Registration Protocol](#)
- [Configuring WS-TX Transactions](#)
- [Configuring Maximum Number of Transactions](#)
- [Configuring Policy Assertions](#)
- [WSDL Generation](#)
- [WSDL Conversion](#)

Notes: These configuration changes are summarized in the `SALTDEPLOY` additions pseudo-schema and WSDL additions pseudo-schema Appendix.

For additional information, see the [Oracle SALT Interoperability Guide](#).

Configuring Transaction Log Device

The GWWS system server must be configured using the transaction log (`TLogDevice`) element (similar to the Oracle Tuxedo or /Domains TLog files). The `TLOGDevice` element is added to the `SALTCONFIG` source file (`SALTDEPLOY`) as shown in [Listing 44](#).

A `TLOGName` element is also added to allow sharing the same TLog device across GWWS instances.

Only one TLog device per Web services Gateway instance is permitted (that is, the transaction log element is a child element of `/Deployment/WSGateway/GWInstance`).

Listing 44 TLOG Element Added to SALTDEPLOY File

```
<Deployment xmlns="http://www.bea.com/Tuxedo/SALTDEPLOY/2007">
  <WSDL>
    ...
  </WSDL>
  <WSGateway>
    <GWInstance id="GW1">
      <TLogDevice location="/app/GWTLOG"/>
      <TLogName id="GW1TLOG"/>
    ...
    </GWInstance>
  </WSGateway>
  ...
</Deployment>
```

Registration Protocol

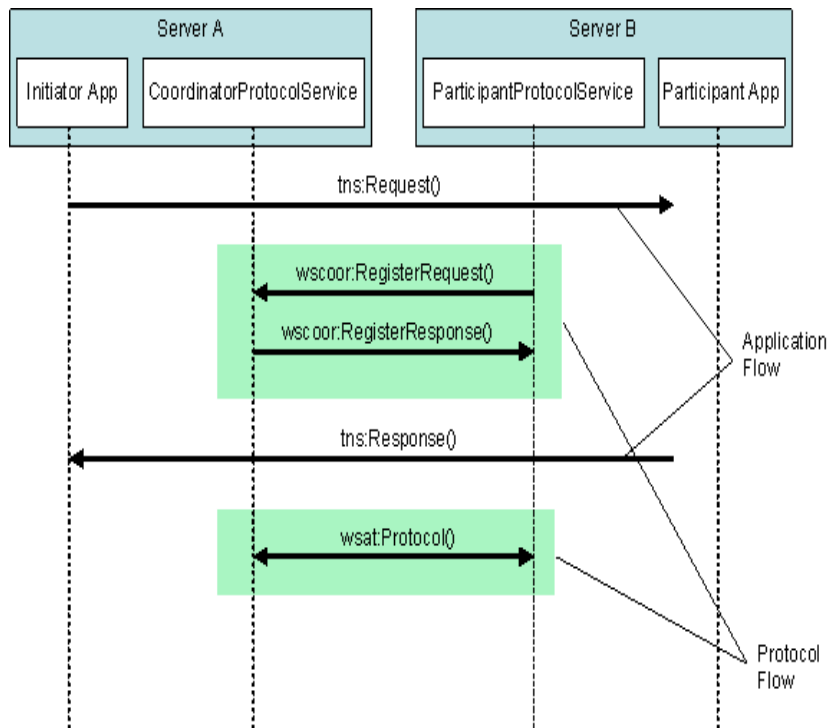
Oracle Tuxedo-based services are registered with a Durable 2PC protocol with coordinators.

When Oracle Tuxedo is the coordinator (outbound direction), the GWWS system server allows either Volatile 2PC or Durable 2PC registration requests and handles them accordingly.

Configuring WS-TX Transactions

Figure 2 illustrates the application and protocol flows of a typical WS-AT context service invocation.

Figure 2 WS-AT Service Invocation



The configuration steps and runtime behavior of the Oracle SALT GWWS gateway are outlined in the following sections (depending on the role of the Oracle Tuxedo domain as shown in Figure 2):

- [Configuring Incoming Transactions](#)
- [Configuring Outbound Transactions](#)

Configuring Incoming Transactions

Oracle Tuxedo services exposed as Web services do not require any specific configuration other than creating a transaction log file and adding it to the GWWS deploy configuration file in order to initiate a local transaction associated with an incoming WS-AT transaction request.

To ensure a transaction can be propagated into an Oracle Tuxedo domain, do the following steps:

1. Ensure that the Oracle Tuxedo service called supports transactions.
2. Configure a transaction log file in the GWWS deployment file. For more information, see [Configuring Transaction Log Device](#).
3. Configure a policy file containing a WS-AT Assertion corresponding to the desired behavior with respect to the external Web Service called. For more information, see [Configuring Policy Assertions](#).
4. Incoming calls containing a `CoordinationContext` element creates an Oracle Tuxedo global transaction.

Error Conditions

Error conditions are handled as follows:

- No log file is configured for the gateway. A `wscoor:InvalidState` fault is sent back to the caller. The `Detail` field contains a corresponding message.
- The target Oracle Tuxedo service does not support transactions. An application fault with a TPETTRAN error code is returned to the caller.
- For all other applications, configuration (such as `TPENOENT`) or system errors are handled the same way that normal (non-transactional) requests are handled.

Configuring Outbound Transactions

In order for Oracle Tuxedo clients to propagate an Oracle Tuxedo global transaction to external Web services, do the following steps:

1. Configure a transaction log file in the GWWS deployment file. For more information, see [Configuring Transaction Log Device](#).

2. Configure a policy file containing a WS-AT Assertion corresponding to the desired behavior with respect to the external Web Service called. For more information, see [Configuring Policy Assertions](#).
3. Depending on the assertion setting and presence of an Oracle Tuxedo transaction context, a `CoordinationContext` element is created and sent in the SOAP header along with the application request.
4. An endpoint reference is automatically generated and sent along with the `CoordinationContext` element for the remote `RegistrationService` element to enlist in the transaction. This step, along with the protocol exchanges (Prepare/Commit or Rollback etc.) is transparent on both sides.

Error Conditions

Error conditions are handled as follows:

- If the remote system does not support transactions and the WS-AT Assertion/transaction context call has *MUST create transaction* semantics, a TPESYSTEM error is returned to the client.
- Errors generated remotely are returned to the Oracle Tuxedo client in the same manner as regular, non-transactional calls. The fault `Reason` and `Detail` fields returned describe the nature of the failure (which is environment dependent).

Configuring Maximum Number of Transactions

The `MaxTran` element allows you to configure the size of the internal transaction table as shown in [Listing 1](#). The default is `MAXGTT`.

Note: The `MaxTran` value is optional. If the configured value is greater than `MAXGTT`, it is ignored and `MAXGTT` is used instead

Listing 1 MaxTran Element

```
<Deployment xmlns="http://www.bea.com/Tuxedo/SALTDEPLOY/2007">
  <WSDF>
    ...
  </WSDF>
  <WSGateway>
```

```
<GWInstance id="GW1">
...
    <MaxTran value="500"/>
...
</GWInstance>
</WSGateway>
...
</Deployment>
```

Configuring Policy Assertions

WS-AT defines a policy assertion that allows requests to indicate whether an operation call **MUST** or **MAY** be made as part of an Atomic Transaction.

Policy.xml File

The `policy.xml` file includes WS-AT policy elements. WS-AT defines the `ATAssertion` element, with an `Optional` attribute, as follows:

`/wsat:ATAssertion/@wsp:Optional="true"` as shown in [Listing 2](#).

Listing 2 Policy.XML ATAssertion Element

```
<?xml version="1.0"?>
<wsp:Policy wsp:Name="TransactionalServicePolicy"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsat="http://docs.oasis-open.org/ws-tx/wsat/2006/06">
  <wsat:ATAssertion wsp:Optional="true"/>
</wsp:Policy>
```

Note: In order to correctly import external WSDLs, the `wsdlcvt` command is modified to generate a `policy.xml` file containing the `ATAssertion` element when one is present in the WSDL. For outbound requests, a `policy.xml` file containing an `ATAssertion` element must be created and properly pointed to in the `SALTDEPLOYSOURCE`.

Inbound Transactions

No particular behavior change will take place at runtime in the case of inbound transactions. The client consuming the WSDL will take the decision based on the configured value and the runtime behavior will follow like for the normal case.

Outbound Transactions

- When an `ATAssertion` with no `Optional=true` is configured, the call must be made in a transaction. If no corresponding XA transaction exists, the WS-TX transaction is initiated but not associated with any Oracle Tuxedo XA transaction. If an XA transaction exists, there is no change in behavior.
- When an `ATAssertion` with `Optional=true` is configured, an outbound transaction context is requested only if a corresponding Oracle Tuxedo XA transaction exists in the context of the call.
- When no `ATAssertion` is configured for this service, the corresponding service call is made outside of any transaction. If a call is made to an external Web service in the context of an Oracle Tuxedo XA transaction, the Web service call will not propagate the transaction.

This allows excluding certain Web service calls from a global transaction, and represents the default for most existing Web services calls (that do not support WS-TX).

WSDL Generation

WSDL generation is enhanced to generate an `ATAssertion` element corresponding to the assertion configured in the policy file for the corresponding service.

WSDL Conversion

For outbound requests, the WSDL conversion tool, `wsdlcvt`, generates a `policy.xml` file containing the `ATAssertion` element when one is present in the processed WSDL. You must properly configure the location of the `policy.xml` file in the `SALTDEPLOY` source.

See Also

- [tmadmin](#)
- [tmloadrepos](#)
- [ubbconfig](#)
- [scaadmin](#)
- [buildscaclient](#)
- [buildscaserver](#)
- [WSDf documentation](#)
- [Oracle SALT Programming Guide](#)
- [Oracle SALT Reference Guide](#)
- [Oracle SALT Interoperability Guide](#)